all rights reversed

HACKER

Season 05 / Issue 07

τεύχος 055 - Áfram Ísland Edition



Τα Version Control Systems και το Git [*]

SQLite, για βάσεις στα γρήγορα

αλγόριθμοι και δυαδική αναζήτηση

ανάπτυξη games και textures που τσακώνονται

τα Version Control Systems και γιατί να ενδιαφερθείτε

το θαυμάσιο Git και πρακτική εξάσκηση με παραδείγματα χρήσης

[*] Ασχοληθείτε τώρα -- μας ευχαριστείτε αργότερα :P

all rights reversed

HACKER


CMYK & WEB
cmykandweb.gr
GRAPHIC DESIGN
design • layout • web

Εκδότης / Διευθυντής	Χρήστος Βαρελάς
Αρχισυντάκτης	Παναγιώτης Βαρελάς
Σύμβουλοι έκδοσης	Νίκος Μουρατίδης, Βούλα Παυλίδου
Σύνταξη	Παναγιώτης Βαρελάς, Χρήστος Βαρελάς, Πάνος Γεωργιάδης, Γιώργος Γιάννου, Μάριος Καραγιαννόπουλος, Γιάννης Κρομμύδας, Πέτρος Κυλαδίτης, Μέσος Παπαδόπουλος (TM), Χρίστος Τόμπρας, @arkoudos
Γραμματική επιμέλεια	Βούλα Παυλίδου
Καλλιτεχνική Επιμέλεια	Πέτρος Φιλιππίδης

Το Το ψηφιακό περιοδικό deltaHacker είναι συνδρομητικό και εκδίδεται από την Parabing Creations. Η Parabing Creations έχει την έδρα της στην Καλαμαριά Θεσσαλονίκης. Ο δικτυακός τόπος του περιοδικού είναι στο http://deltahacker.gr. Το email επικοινωνίας είναι το talk2us@deltahacker.gr και ειδικά για τις συνδρομές είναι το subscriptions@deltahacker.gr. Οι γενικές πληροφορίες για τις συνδρομές παρατίθενται στο http://deltahacker.gr/pdf12order και από την ίδια σελίδα γίνονται οι παραγγελίες συνδρομών.

GNOMElightenment!

του Παναγιώτη Βαρελά

Φίλες και φίλοι,

- Έχουν περάσει δεκαπέντε χρόνια από το 2001.
- Εκείνο το έτος αποτέλεσε ορόσημο στην τεχνολογική μου σταδιοδρομία, καθώς βρήκα το κουράγιο να εγκαταστήσω για πρώτη φορά το Linux.
- Μη βιαστείτε να με χαρακτηρίσετε ως κακεντρεχή ή κάτι τέτοιο. Αγαπώ κι εγώ το Linux, αλλά η εγκατάστασή του ήταν ιδιαίτερα απαιτητική τότε. Ξεχάστε τους οδηγούς εγκατάστασης και τα εύχρηστα GUI, την αυτόματη διαίρεση των δίσκων σε κατατμήσεις, την αυτόματη ρύθμιση του boot manager και την αυτόματη επιλογή πακέτων. Ένα μικρό λαθάκι ήταν αρκετό για να καταλήξεις μ' ένα σύστημα που δεν ξεκινά καθόλου ή, ακόμα χειρότερα, μ' έναν ισοπεδωμένο σκληρό δίσκο.
- Η πρόσβαση στο Internet ήταν απελπιστικά αργή (άσχετα με το Linux, ναι) και οι πηγές για τη λήψη βοηθείας ελάχιστες. Ρωτούσες το πρωί σε κάποιο forum κι αν ήσουν τυχερός σου απαντούσαν το απόγευμα. Κι αν κατάφερνες να εντοπίσεις κάποιο καλογραμμένο κι επεξηγηματικό κείμενο, έπρεπε να το τυπώσεις: smartphone και tablet για να το διαβάσεις από εκεί, δεν υπήρχαν.
- Κάπως έτσι άρχισα να ταλαιπωρώ τους δύο 40ρηδες δίσκους του υπολογιστή μου. Έβγαζα τη μία διανομή κι έβαζα την άλλη, ενώ πολλές φορές εγκαθιστούσα την ίδια διανομή ξανά και ξανά, μόνο και μόνο γιατί η προηγούμενη εγκατάσταση ήταν προβληματική.
- Αυτή η δουλειά κράτησε αρκετά χρόνια, με πολύ μικρά διαλείμματα κατά τα οποία εγκαθιστούσα Windows. Τα κίνητρα γι' αυτή την προδοσία ήταν πολύ συγκεκριμένα: Unreal Tournament και Half-Life: Counter-Strike. Παιχνίδια του 1999 και 2000, αντίστοιχα, που είχαν προλάβει να γίνουν κλασικά!
- Μέσα στο γενικότερο χαμό που επικρατούσε στους δίσκους του υπολογιστή, μόνο ένα πράγμα έμενε σταθερό: Το περιβάλλον GNOME και οι ρυθμίσεις του, που φρόντιζα να μη χάνονται ποτέ. Αυτή η εκτίμηση, όμως, δεν αναπτύχθηκε από την πρώτη στιγμή που χρησιμοποίησα Linux.

/var/log/messages

- Μη νομίζετε ότι κάθε φορά που άλλαζα διανομή ενδιαφερόμουν για τις αρχιτεκτονικές ιδιαιτερότητες της καθεμίας. Τον πρώτο καιρό μεταπηδούσα από το ένα σύστημα στο άλλο, μόνο και μόνο γιατί αυτός ήταν ο ευκολότερος τρόπος να δοκιμάσω κάποιον άλλο window manager. Το κατέβασμα μεγάλων εφαρμογών που απαρτίζονταν από πολλά πακέτα δεν ήταν πάντα εφικτό, ενώ το κατέβασμα ενός CD ISO από τα μηχανήματα της σχολής ήταν παιχνιδάκι. Σου έδινε και τη δικαιολογία για να πιεις έναν ακόμα καφέ.
- Free Virtual Window Manager: αυτό ήθελα να το γράψω, είναι σχεδόν συγκινητικό.
- Αφού προσπέρασα τον FVWM, ξόδεψα αρκετά βράδια μπροστά από τον Window Maker. Κι όταν το ομολογουμένως έξυπνο όνομα έπαψε να με εντυπωσιάζει, πέρασα μερικές μέρες ρυθμίζοντας τη συμπεριφορά ενός drawer στον AfterStep.
- Μετά δοκίμασα το BlackBox, το OpenBox και το Enlightenment, του οποίου την εξέλιξη παρακολουθώ ακόμα. Τι, δεν με πιστεύετε; Το έκανα πριν από λίγο, δεν μετράει;
- Πάντως ο δρομέας του terminology –του terminal emulator που συνοδεύει τον εφετζίδικο Enlightenment–, συνιστά τη μεγαλύτερη ποζεριά που έχω συναντήσει στο χώρο των υπολογιστών. Είναι κάτι σαν τα κουτιά με καλύμματα από Plexiglas και φωτισμό με μπλε LED, μόνο που είναι απείρως πιο καλαίσθητος.
- Όταν αποφάσισα να σταματήσω το παιχνίδι με τους window managers, στράφηκα στο KDE και στο GNOME. Αυτά τα δύο συστήματα ήταν ογκώδη και δυσκίνητα σε σχέση με οτιδήποτε είχα δει ως τότε. Προσέφεραν όμως όλες τις λειτουργίες ενός σύγχρονου περιβάλλοντος και, κυρίως, πάρα πολλές ευκολίες. Για να αλλάξεις κάτι στο περιβάλλον, π.χ., ένα link στο κεντρικό μενού, αρκούσαν μερικά κλικ και δεν χρειαζόταν να εντρυφήσεις στη δομή κάποιου (κρυφού) αρχείου ρυθμίσεων.
- Μετά απ' όλη αυτή την περιπλάνηση, η τελική απόφαση ελήφθη γρήγορα: Οι σκέτοι window managers και το KDE απορρίπτονται. Για διαφορετικούς λόγους, βεβαίως. Οι πρώτοι συνιστούσαν ένα εξαιρετικά φτωχό περιβάλλον, ενώ το KDE ήταν φορτωμένο με περιττά μπιχλιμπίδια. Αφήστε το άλλο! Ορισμένα themes του KDE προσπαθούσαν αγωνιωδώς να αντιγράψουν το περιβάλλον των Windows, γεγονός που μου προκαλούσε απέχθεια. Ναι, η αντιπάθεια προς τη Microsoft ήταν ήδη στη μόδα :)

- Το KDE προσέφερε εκατοντάδες ρυθμίσεις. Μπορούσες να αλλάξεις θέση, μέγεθος και χρώμα σε οτιδήποτε εμφανιζόταν στην οθόνη. Σε άφηνε να τροποποιήσεις τα πάντα. Κάποιοι ισχυρίζονταν ότι αυτό το χαρακτηριστικό προσέφερε στο χρήστη την απόλυτη ελευθερία, που για τους ίδιους κάποιους, αποτελούσε το κύριο ζητούμενο από ένα ελεύθερο λειτουργικό σύστημα. Εγώ, από την άλλη, πίστευα ότι έχουν μπερδέψει τις διαφορετικές εννοιολογικές αποχρώσεις της λέξης "ελεύθερος" και, θέλοντας να παίξω με τη σύγχυσή τους, σημείωνα το εξής: οι αμέτρητες, κυριολεκτικά, ρυθμίσεις, μοιάζουν με λαβύρινθο – με μια περίτεχνη φυλακή!
- Το GNOME πάλι ακολουθούσε εντελώς διαφορετική φιλοσοφία. Είχε σαφή άποψη για το στήσιμο του περιβάλλοντος και δεν φοβόταν να την επιβάλει. Ταυτόχρονα προσέφερε ένα ικανοποιητικό περιθώριο παραμετροποίησης, ενώ ήταν και αισθητά πιο ελαφρύ έναντι του KDE.
- Για πολλά χρόνια το KDE και το GNOME δεν είχαν φίλους. Είχαν οπαδούς που σύχναζαν σε fora και σε κανάλια του IRC, έχοντας πάντα τη διάθεση να εμπλακούν σε μια ατελείωτη αντιπαράθεση. Ατελείωτη και κυρίως ανούσια, αφού το GNOME ήταν ξεκάθαρα ανώτερο :P
- Ε, λοιπόν, δεκαπέντε χρόνια μετά, απέκτησα έναν ακόμη λόγο για ν' αγαπώ το GNOME.
- Πριν από μερικούς μήνες ξεκίνησα ένα σύνθετο project, άσχετο με το περιοδικό. Περιλάμβανε την κατασκευή διαφόρων κυκλωμάτων, τον προγραμματισμό ενός AVR και την ανάπτυξη ενός παραθυρικού προγράμματος για το Raspberry Pi.
- Το ξεκίνησα μαζί μ' έναν φίλο ο οποίος έχει καθαρά προγραμματιστικό background, σε αντίθεση μ' εμένα που έχω εντρυφήσει περισσότερο στο hardware και στα ηλεκτρονικά. Όταν ωστόσο ολοκληρώθηκαν τα απαραίτητα κυκλώματα, καταλήξαμε να γράφουμε κώδικα και οι δύο.
- Στο Raspberry Pi εγκαταστήσαμε το Raspbian, ενώ για την ανάπτυξη της εφαρμογής επιλέξαμε τη γλώσσα C και την εργαλειοθήκη GTK+. Τίμια πράγματα!
- Για τη συγγραφή του κώδικα χρησιμοποιήσαμε το Geany. Μέχρι πριν από λίγους μήνες αγνοούσα την ύπαρξή του. Σήμερα; Σήμερα τολμώ να πω ότι είναι ο αγαπημένος μου editor. Θα μπορούσα ν' αραδιάσω δεκάδες λόγους που μ' έκαναν να το αγαπήσω, αλλά θα σας πω μόνον έναν: Κρατάς πατημέ-

νο το [CTRL], κάνεις κλικ επάνω στο όνομα μιας συνάρτησης κι ως δια μαγείας μεταφέρεσαι στο αρχείο και στο σημείο όπου βρίσκεται ο κώδικάς της. Ανεκτίμητη ευκολία.

- Μη φανταστείτε ότι δουλεύουμε στο ίδιο εργαστήριο ή με βάρδιες. Ο καθένας εργάζεται από το χώρο του και γράφει κώδικα οποτεδήποτε του καπνίσει, σε οποιοδήποτε αρχείο και σε όποια θέση του αρχείου θελήσει. Ο αχταρμάς που προκύπτει με αυτόν τον τρόπο μετατρέπεται σε λειτουργικό πρόγραμμα με τη βοήθεια του Git. Θεϊκό εργαλείο. Αν έπρεπε να το διαφημίσω στην τηλεόραση, θα έλεγα ότι έχει εγκέφαλο.
- Στις ώρες του μεγάλου πανικού έχει τύχει να προσθέτουμε δεκάδες γραμμές ο καθένας, σε σχεδόν τυχαία σημεία μέσα στο ίδιο αρχείο. Το Git δεν μάσησε ποτέ. Κάποιες άλλες φορές (ευτυχώς λιγότερες) έτυχε να τροποποιήσουμε και οι δύο την ίδια συνάρτηση. Το Git δεν μάσησε ούτε και τότε. Θα έλεγε κανείς ότι ξέρει τι προσπαθούμε να γράψουμε, καλύτερα από εμάς τους ίδιους.
- Για τη σχεδίαση του GUI χρησιμοποιήσαμε το Glade. Πολλές φορές, ωστόσο, περιοριστήκαμε στο Geany. Το αρχείο που περιγράφει την όψη του προγράμματος είναι ένα απλό XML.
- Για μένα, τον hardware guy, η γνωριμία μ' όλα αυτά τα εργαλεία αποτέλεσε μια συναρπαστική εμπειρία. Αυτό που εκτίμησα περισσότερο, όμως, ήταν η βιβλιοθήκη Glib.
- Προσέξτε, δεν αναφέρομαι στην Glibc – GNU C standard library. Αυτή περιλαμβάνει wrapper για τα system calls που προσφέρει ο πυρήνας και για κάποιες άλλες θεμελιώδεις εργασίες.
- Η Glib –χωρίς to "c" στα δεξιά– αποτελεί το υπόβαθρο του GNOME. Ξεκίνησε σαν τμήμα του GTK+, αλλά πολύ γρήγορα ο κώδικάς της διαχωρίστηκε και σχημάτισε μια αυτοτελή βιβλιοθήκη. Κι όταν λέμε αυτοτελή, κυριολεκτούμε. Η Glib χρησιμοποιείται απαραιτήτως σε όλα τα προγράμματα που στηρίζονται στο GTK+, αλλά μπορεί να χρησιμοποιηθεί και σε εφαρμογές της κονσόλας.
- Ανάμεσα στις αμέτρητες ευκολίες που προσφέρει, διαθέτει μια σειρά από βασικά data types. Στην πράξη, δεν διαφέρουν από τους βασικούς τύπους δεδομένων της C, μόνο που τα ονόματα και ο χειρισμός τους είναι πολύ πιο ομοιόμορφος. Αυτό, θα πείτε, λύνεται με μερικά typedef και δεν αποτελεί

ουσιαστικό λόγο για να προτιμήσουμε μια βιβλιοθήκη. Ωστόσο, τα ενσωματωμένα data types αναδεικνύουν την ομοιομορφία και την κομψότητα που διέπουν ολόκληρη τη βιβλιοθήκη.

- Η Glib διαθέτει την υποδομή για το σχηματισμό ενός "main loop" και το χρονοπρογραμματισμό διαφόρων εργασιών. Αν αυτές οι λειτουργίες είναι αρκετά υψηλού επιπέδου για τα γούστα σας, στο οπλοστάσιο της Glib θα βρείτε μηχανισμούς για τη δημιουργία multi-threaded εφαρμογών καθώς κι αρκετούς ακόμα, που καταπιάνονται με θεμελιώδεις εργασίες. Πάρτε για παράδειγμα τα GIOChannel, που επιτρέπουν τον εύκολο και ομοιόμορφο χειρισμό αρχείων, pipes και sockets.

- Δυστυχώς, μέσα από αυτές τις γραμμές είναι πρακτικά αδύνατο να σας πείσω. Για να εκτιμήσει κανείς τη Glib πρέπει να τη χρησιμοποιήσει. Μια σειρά άρθρων πάνω στο θέμα θα ήταν καλή ιδέα. Θα περιμένω να δροσίσει λίγο κι αν είναι θα βάλω μπρος.

- Δεν ξέρω αν θα συμβεί το ίδιο και μ' εσάς, αλλά μέσα από αυτή τη γνωριμία εγώ τουλάχιστον βρήκα έναν πρόσθετο λόγο για να υποστηρίζω το GNOME. Δεν αποκλείεται να πάθαινα το ίδιο και με κάποια άλλη βιβλιοθήκη. Η Glib, όμως, είναι μοναδική γιατί αποτελεί θεμελιώδες συστατικό του GNOME. Ναι, συμφωνώ. Αυτό το σχήμα είναι κυκλικό αλλά...

- GNOME!

POST CARD

THE ADDRESS TO BE WRITTEN ON THIS SIDE



ουέλ, μέλ

Αγαπητοί μου,

Συνειδητοποιώ πως είμαι λίγο πεσμένος και ανήσυχος αυτές τις μέρες.

Δεν είναι κάτι απτό και συγκεκριμένο, όχι. Πιο πολύ μια αίσθηση πως τα πράγματα έχουν ξεφύγει από τον έλεγχο και μια αδυναμία να εικάσει κανείς προς τα πού θα πάνε.

Μην με παρεξηγάτε, αγαπητοί μου, ούτε μέντιουμ είμαι, ούτε υπήρξα μέντιουμ στο παρελθόν. Απλά θα ήθελα να έχω μιαν αίσθηση ότι τα πράγματα προχωρούν προς κάπου, και να έχω μιαν ασαφή, έστω, εικόνα γι' αυτό το "κάπου".

Δεν μιλάω μόνο για την Ελλάδα. Η, μάλλον, δεν μιλάω καν για την Ελλάδα.

Για την Ελλάδα ξέρω μια χαρά πού πάνε τα πράγματα. Πάνε κατά διαόλου. Κατά διαόλου, και με τα χίλια. Πώς λέμε "φως στην άκρη του τούνελ"; Ε, that's not it.

Μιλούσαμε χτες το απόγευμα στο σκάιπ με τον εν Λονδίνοις ανιψιό. Έχει πάει αυτές τις μέρες από εκεί η Μαιρούλα, για "εκδρομή", και μας μιλούσανε χτες, λίγο πριν βγούνε για βόλτα.

Λέγαμε διάφορα. Η μάλλον *λέγανε* διάφορα. Εγώ ήμουν από δίπλα κι άκουγα, αλλά χωρίς να συμμετέχω. Λέγανε για τον καιρό, για το θεατρικό με τον Χάρυ Πότερ που ανοίγει από μέρα σε μέρα (ή μήπως έχει ήδη ανοίξει; --θα σας γελάσω), για το νέο Δήμαρχο, για την βροχή, για τον παλιό Δήμαρχο, για τα καινούργια διώροφα λεωφορεία, για τη βροχή, για τα παλαιότερα διώροφα λεωφορεία, για τη βροχή, και τα λοιπά, και τα λοιπά.

"Τι κάνετε, ετοιμάζεστε για BREXIT;" τον ρωτάει σε κάποια στιγμή η κυρία Καίτη.

Δεν είδα την έκφραση του ανιψιού, αλλά άκουσα τη σιωπή του.

Να το, αυτό που σας έλεγα στην αρχή. BREXIT. Ένα από τα πράγματα που με ανησυχούν. Το ενδεχόμενο για ένα BREXIT.

Όλοι λέμε διάφορα. Όλοι γνωρίζουν. Όλοι έχουν γνώμη. Όλοι το προεξοφλούν. Όλοι επιχαίρουν. Όλοι το ερμηνεύουν.

Για πάτε μια βόλτα από την Εβγα του κυρίου Κώστα και θα με θυμηθείτε. Το BREXIT, κατ' αυτούς, είναι ράπισμα κατά της Μέρκελ.

Φαντασιώνονται "ράπισμα", αγαπητοί μου, και χαίρονται.

Μη λέω πολλά. Εδώ που τα λέμε είμαι σίγουρος πως κι εσείς χαίρεστε. Κάποιοι από εσάς έστω, από εσάς που μου κάνετε την τιμή να διαβάζετε τις επιστολές αυτές.

Χαίρεστε, και δεν είστε οι μόνοι. Χαίρεστε γιατί θα έχει πλάκα. Γιατί θα τρέχει η Μέρκελ και δεν θα φτάνει. Θα προσπαθεί ο Ολάντ να μαζέψει τα ασυμμάζευτα. Γιατί θα παραιτηθεί ο Κάμερον. Θα τσαντιστούν οι Αμερικάνοι.

Κου λου που, κου λου που. Θα γίνει χαμός. Θα τιμωρηθεί η Ευρώπη. Γι αυτό χαίρεστε.

Δεν ξέρω αν έχω δίκιο σ' αυτή μου την εκτίμηση, μπορεί και να μην χαί-

ρεστε καθόλου. Αλλά δεν έχει σημασία. Δεν είναι προσωπικό το θέμα. Αυτό που έχει σημασία είναι πως κάποιοι χαίρονται.

Οι περισσότεροι.

Χαίρονται και γελούν, οι μωροί, κ'αν τι μη γελοίον ει.

Λοιπόν παιδιά, θα το πω καθαρά.

Όσοι χαίρονται για το ενδεχόμενο ενός BREXIT είναι βαθιά νυχτωμένοι. Είναι εκτός θέματος. Δεν ξέρουν τι λένε. Το μυαλό τους και μια λίρα, που θα έλεγαν και οι παλιοί.

Προσέξτε, μιλάω χωρίς να με νοιάζουν καθόλου τα συμφέροντα της Γηραιάς Αλβιώνος. Ή τα συμφέροντα όσων έχουν ξεμείνει στις Αγγλίες, όπως ο ανιψιός, και δεν θα ξέρουν πού να κρυφτούν σε περίπτωση που πράγματι νικήσει το BREXIT.

Καθόλου. Η βασική μου ανησυχία έχει να κάνει με το τι θα απογίνει η Ευρώπη.

Αγαπητοί μου,

Η Ευρώπη, ειρηνική όπως την έχουμε συνηθίσει τα τελευταία 70 χρόνια, είναι χτισμένη πάνω από τρεις ορόφους υπόγεια γεμάτα με μπαρούτι. Δεν έχει θεμέλια, όχι. Μπουντρούμια με μπαρούτι έχει. Κατακόμβες.

Κλειδωμένα, δε λέω, αλλά τα μπουντρούμια είναι εκεί. Γεμάτα μπαρούτι. Περιμένουν, γιατί δεν έχουν και κάτι καλύτερο να κάνουν. Δεν βιάζονται -- γιατί να βιαστούν άλλωστε;

Οι άνθρωποι όμως βιάζονται. Όχι μόνο βιάζονται, αλλά και ξεχνούν. Θυμούνται επιλεκτικά. Ή, για να το πω καλύτερα, *επειδή* ξεχνούν --ή θυμούνται επιλεκτικά-- γι αυτό βιάζονται.

Παιδιά, αντίθετα με όσα θα ακούσετε να σας λένε στην Εβγα του Κυρίου Κώστα, η Ευρωπαϊκή Ένωση *δεν* φτιάχτηκε για να επιβάλει τα συμφέροντα του μεγάλου κεφαλαίου.

Δεν φτιάχτηκε για να επιβάλει τα συμφέροντα της Γερμανίας.

Δεν φτιάχτηκε για να χτυπήσει τα λαϊκά στρώματα.

Δεν φτιάχτηκε για να ταπεινώσει τον ελληνισμό ή για να μας μετατρέψει όλους σε αμερικανάκια.

Δεν είναι πλεκτάνη του διεθνούς σιωνισμού.

Η Ευρωπαϊκή Ένωση ήταν προϊόν της συνειδητής προσπάθειας όσων επέζησαν από το Δεύτερο Παγκόσμιο Πόλεμο, να χώσουν κάποια από τα βαρέλια με το μπαρούτι που περίσσεψε όσο πιο βαθιά στο μπουντρούμι γίνεται και μετά να κλειδώσουν την πόρτα ελπίζοντας να μη γίνει καμιά στραβή.

Αυτό ήταν. Τελεία και παύλα. Είπα και ελάλησα.

Θα μου πείτε, τι εννοείς είπες και ελάλησες; Δεν τη βλέπουμε λες τη φτωχοποίηση (sic) της Ελλάδας μας; Εσένα περιμέναμε για να δούμε τους εκβιασμούς; Δεν την βλέπουμε την επικυριαρχία της Μέρκελ;

Κι αρχίζει η κολοκυθιά.

Όμως το θέμα είναι σοβαρό, κι επειδή ο χώρος είναι περιορισμένος θα απαντήσω επιγραμματικά.

Έχουμε δύο ζητήματα εδώ, που το ένα μπλέκει με το άλλο.

Το ένα ζήτημα είναι η ΕΕ σήμερα και οι ζυμώσεις που λαμβάνουν χώρα στα κέντρα εξουσίας της: Γερμανία εναντίον Γαλλίας, Αγγλία εναντίον Γαλλογερμανίας, Βορράς εναντίον Νότου κ.λπ.

Το άλλο είναι η ίδια η Ευρώπη (με τη γεωγραφική έννοια του όρου) και οι εθνικιστικές κι άλλες εντάσεις στο εσωτερικό της, που στο παρελθόν έχουν ήδη ανάψει το φυτίλι για δύο παγκόσμιους πολέμους.

Τα δύο ζητήματα μοιάζουν μεταξύ τους αλλά δεν ταυτίζονται.

Η ΕΕ φτιάχτηκε για να αποτρέψει περαιτέρω



αναφλέξεις σαν αυτές που έδωσαν τους πολέμους. Δεν φτιάχτηκε για να δώσει την ευκαιρία στην Γερμανία να πετύχει αυτό που δεν πέτυχε με τα όπλα.

Ο συλλογισμός είναι κυκλικός και ως εκ τούτου λανθασμένος.

Post hoc ergo propter hoc. (Γκουγκλ ιτ!)

Το ότι η Γερμανία σήμερα είναι στην θέση που είναι μπορεί να μας αρέσει ή να μη μας αρέσει, αλλά είναι μυωπικό να νομίζουμε πως ένα "ράπισμα" τύπου BREXIT θα έχει κάποιο άλλο αποτέλεσμα από το να συμβάλει στο γενικό χάος.

Το BREXIT θα οδηγήσει με μαθηματική ακρίβεια στην κατάρρευση της ευρωπαϊκής ένωσης με τη μορφή που έχει σήμερα. Και η κατάρρευση της ευρωπαϊκής ένωσης θα οδηγήσει με μαθηματική ακρίβεια σε μείζονα ενδοευρωπαϊκή σύγκρουση.

Σένγκεν; Ευρωπαϊκό όραμα; Α, καλά.

Αλλά όχι αύριο ή μεθαύριο. Σε 15 με 20 χρόνια.

Remember: You read it here first.

Και κάτι ακόμα, ως τροφή για σκέψη.

Ας πάρουμε οποιαδήποτε διαδικασία όπου λαμβάνει χώρα κάποια ζύμωση.

Κάποια στιγμή η ζύμωση οδηγεί σε πράξεις. Λαμβάνονται αποφάσεις. Γίνονται κινήσεις.

Κάθε μα κάθε πράξη, απόφαση, κίνηση, είναι προϊόν ενός υπερπολύπλοκου συμβιβασμού.

Όταν πλέον λαμβάνεται η απόφαση, τα επιμέρους συμφέροντα, αυτά επί των οποίων έγινε ο συμβιβασμός, ξεχνιούνται. Το reverse engineering είναι αδύνατο. Οι προηγούμενες επί μέρους ζυμώσεις έχουν πλέον περάσει τον ορίζοντα των γεγονότων και είναι αδύνατο να ανακληθούν.

Ναι, η πληροφορία χάνεται. Ναι. Μην ακούτε τι λέει η κβαντομηχανική. Οι ανθρώπινες κοινωνίες δεν συμμορφώνονται στα μοντέλα της κβαντομηχανικής.

Κάθε προσπάθεια αναδρομικής ερμηνείας των αρχικών συμβιβασμών και ζυμώσεων είναι ακριβώς αυτό. Αναδρομική ερμηνεία. Βασίζεται σε εικασίες και είναι αυθαίρετη.

Προσοχή, "αυθαίρετη", δεν σημαίνει πως είναι κατ' ανάγκη λάθος. Όχι. Σημαίνει ότι δεν προκύπτει από τα δεδομένα, απλά "κουμπώνει" σε αυτά.

Δεν υπάρχει *τίποτα* στα εκάστοτε τρέχοντα δεδομένα που θα μπορούσε να με ωθήσει να σκεφτώ πως μια απόφαση τάδε ελήφθη για τον άλφα ή για τον βήτα λόγο. Το μόνο που μπορούν να μου δείξουν τα εκάστοτε τρέχοντα δεδομένα είναι αν κάποια εικασία μου στέκει ή δεν στέκει (δηλαδή "κουμπώνει" ή δεν "κουμπώνει").

Τίποτε άλλο.

Να το πω και πιο τεχνικά: Το γεγονός ότι είμαι σε θέση να φτιάξω την αναδρομική ερμηνεία μιας οποιασδήποτε σημερινής κατάστασης, λαμβάνοντας υπόψη μου όλα μα όλα τα τρέχοντα δεδομένα, *δεν* ισοδυναμεί με το ότι έχω κατανοήσει την αιτιοκρατική αλυσίδα που έχει οδηγήσει στη σημερινή κατάσταση. Δεν σημαίνει καν πως υπάρχει αιτιοκρατική αλυσίδα.

Η ανθρώπινη ιστορία είναι ερμηνεύσιμη αλλά όχι ντετερμινιστική. Αντίθετα με όσα έλεγε ο θείος Κάρολος.

Ας το κρατήσουμε αυτό.

Θα επανέλθω.

Σας ασπάζομαι,

Θείος Ακάκιος



CMYK & WEB
cmykandweb.gr
GRAPHIC DESIGN
design • layout • web

Skill: Beginner
Tags: Algorithms, binary search, performance, complexity

Γνωριμία με τους αλγορίθμους:
Δυαδική Αναζήτηση

Ένας αλγόριθμος δεν είναι τίποτε άλλο παρά ένα σύνολο οδηγιών, οι οποίες στοχεύουν στην επίλυση ενός προβλήματος. Τι είδους προβλήματος; Οποιουδήποτε είδους. Πόση ώρα πρέπει να βράσω το κρέας; Πόσο νερό θα χρειαστώ; Τι μπαχαρικά πρέπει να προσθέσω στην κατσαρόλα; Ερωτήματα σαν τα προαναφερθέντα αφορούν σε προβλήματα που λύνονται με συνταγές μαγειρικής, οι οποίες, ουσιαστικά, είναι αλγόριθμοι.

του Πάνου Γεωργιάδη

Αν ξέρουμε μια συνταγή, τότε μπορούμε να μαγειρέψουμε. Ομοίως: Γνωρίζοντας έναν αλγόριθμο, μπορούμε να λύσουμε το αντίστοιχο πρόβλημα. Στον κόσμο της πληροφορικής υπάρχουν ορισμένα προβλήματα, τα οποία καλείται ν' αντιμετωπίσει κάθε προγραμματιστής. Αν και οι σχετικοί αλγόριθμοι είναι ήδη γνωστοί και πολύ καλά μελετημένοι, οι μαθητευόμενοι προγραμματιστές οφείλουν να τους γνωρίζουν – τουλάχιστον κάποιους εξ αυτών. Παραθέτουμε τρία μόνο παραδείγματα προβλημάτων, για τα οποία υπάρχουν γνωστοί αλγόριθμοι:

- Πώς κάνουμε μια αναζήτηση πιο γρήγορη και πιο αποτελεσματική;
- Πώς μπορεί να υπολογίσει ένα GPS τη βέλτιστη διαδρομή προς κάποιον προορισμό;
- Πώς υλοποιούμε το υποσύστημα Τεχνητής Νοημοσύνης ενός προγράμματος που παίζει σκάκι;

Για κάθε πρόβλημα που έχει ήδη επινοηθεί αλγόριθμος, η ιστορία δεν σταματά εκεί. Οι ερευνητές, βλέπετε, επιθυμούν κι αναλύουν την ταχύτητα των αλγορίθμων. Όπως και να το κάνουμε, είναι ένα πράγμα να γνωρίζουμε κάποιον αλγόριθμο ο οποίος φέρνει αποτελέσματα σε 12 μήνες, κι άλλο να επινοήσουμε αλγόριθμο που για το ίδιο πρόβλημα φέρνει αποτελέσματα σε 2 ώρες. Σε αντίθεση με τους Μαθηματικούς, οι οποίοι μένουν απόλυτα ικανοποιημένοι όταν γνωρίζουν πως ένα πρόβλημα *έχει* λύση, οι επιστήμονες της Πληροφορικής θέλουν να γνωρίζουν και πόσο *γρήγορα* φτάνουν σε μια λύση. Οι αναλύσεις και οι συζητήσεις περί ζητημάτων ύπαρξης κι αποδοτικότητας δεν εξαντλούνται, αλλά δεν αποτελούν το κεντρικό θέμα του άρθρου που τώρα διαβάζετε. Προς το παρόν, σημειώνουμε μόνο ότι για την ανάλυση της απόδοσης ή αλλιώς της πολυπλοκότητας των αλγορίθμων, στην επιστήμη της Πληροφορικής χρησιμοποιείται ο συμβολισμός του "big O". Περισσότερα επ' αυτού θα πούμε λίγο αργότερα.

Αξίζει να μελετάμε τους αλγορίθμους;

Η απάντηση είναι καταφατική, ασχέτως αν είμαστε επαγγελματίες προγραμματιστές ή ασχολούμαστε με τον προγραμματισμό ως χομπίστες. Σε κάθε περίπτωση, πολλές φορές οφείλουμε να έχουμε κατά νου την έννοια της επίδοσης (performance). Αυτό δεν σημαίνει ότι χρειάζεται ν' ανακαλύπτουμε ξανά και ξανά τον τροχό, αφού όλοι οι βασικοί αλγόριθμοι είναι γραμμένοι από κάποιους άλλους και μάλιστα σε διάφορες γλώσσες προγραμματισμού. Πρέπει, ωστόσο, να έχουμε την ικανότητα ν' απαντάμε σε ερωτήματα όπως τα ακόλουθα: Ποιον αλγόριθμο χρειάζομαι για το δεδομένο πρόβλημα; Αξίζει να καταφύγω στο Quick Sort ή να στραφώ στο Merge Sort; Θα πρέπει να χρησιμοποιήσω array ή list; Μόνον όταν είμαστε σε θέση να κατανοούμε τα πλεονεκτήματα και τα μειονεκτήματα των αλγορίθμων, θα επιλέγουμε σωστά αλγορίθμους ή/και δομές δεδομένων.

Μην ξεχνάτε άλλωστε ότι τα παραπάνω αποτελούν προαπαιτούμενα για κάποιον που θέλει να εργαστεί ως προγραμματιστής. Πριν κάνετε λοιπόν αίτηση στη Google, οφείλετε να γνωρίζετε ζητήματα που αφορούν στους αλγορίθμους πάρα, μα πάρα πολύ καλά.

Τι χρειάζεται να ξέρει κανείς για ν' ασχοληθεί;

Αν ακούτε για αλγορίθμους και τρέχετε μακρυά λόγω των μαθηματικών που βρίσκονται από πίσω, τότε βρίσκεστε στο σωστό μέρος. Για την κατανόηση όσων συζητάμε στο πλαίσιο του παρόντος άρθρου χρειάζονται μόνο βασικές γνώσεις Άλγεβρας και Συναρτήσεων, όπως τις μαθαίνουμε στο Γυμνάσιο. Για παράδειγμα, έστω ότι έχουμε τη συνάρτηση f(x) = 5*x. Αν αμέσως καταλαβαίνετε ότι f(10)=50, τότε άνετα μπορείτε να συνεχίσετε με την ανάγνωση του άρθρου. Επίσης, καλό θα ήταν να γνωρίζει κανείς μία γλώσσα προγραμματισμού. Ξεκινάτε τώρα και δεν μπορείτε ν' αποφασίσετε με ποια πρέπει να καταπιαστείτε; Σας προτείνουμε την Python, καθώς αποτελεί εξαίσια επιλογή για αρχαρίους. Και τώρα που γνωρίζετε τα εφόδια που πρέπει να έχετε, μπορείτε να συνεχίσετε ακάθεκτοι με το υπόλοιπο του άρθρου. Σε λίγο θα υλοποιήσουμε τον πρώτο μας αλγόριθμο, ο οποίος ακούει στο όνομα binary search.

Δυαδική Αναζήτηση

Θυμόσαστε τους παλιούς τηλεφωνικούς καταλόγους; Ναι, εκείνο το μεγάλο χοντρό βιβλίο που μας μοίραζαν κάθε χρονιά στην είσοδο της πολυκατοικίας, κι έγραφε πάνω του "Χρυσός Οδηγός". Φανταστείτε τώρα ότι δεν έχετε Internet και χρειάζεται να καταφύγετε σ' αυτό το βιβλίο, ώστε να βρείτε το τηλέφωνο μιας συγκεκριμένης επιχείρησης το όνομα της οποίας αρχίζει από το γράμμα "Π". Σαφώς, μπορεί κανείς να ξεκινήσει και να ξεφυλλίζει από την αρχή, μέχρι να φτάσει στο "Π". Νομίζουμε ωστόσο ότι οι περισσότεροι θ' άνοιγαν τυχαία τον τηλεφωνικό κατάλογο, κάπου στη μέση, καθώς το "Π" βρίσκεται στη μέση περίπου του αλφάβητου.

Όσοι δεν είχατε τη χαρά να χρησιμοποιήσετε το Χρυσό Οδηγό, μάλλον είχατε παρόμοια εμπειρία με κάποιο έντυπο λεξικό. Ας υποθέσουμε ότι κάποιος ψάχνει μια λέξη στ' αγγλικά, η οποία αρχίζει από το γράμμα "O". Ξανά, υποθέτουμε ότι ο περισσότερος κόσμος θ' άνοιγε το λεξικό περίπου στη μέση. Στη συνέχεια θα ξεφύλλιζε μπρος-πίσω, έως ότου βρει το "O" και μετά τη λέξη που ψάχνει.

Αρκετά όμως με το παρελθόν και με τις ξεπερασμένες μεθόδους αναζήτησης. Επιστροφή στο σήμερα, όπου έχετε ανοίξει τον web browser και είστε έτοιμοι να κάνετε login στο Facebook. Πληκτρολογείτε, λοιπόν, τα στοιχεία σας κι αμέσως μετά κάνετε κλικ στο κουμπί "login". Εκείνη τη στιγμή, το Facebook καλείται να πιστοποιήσει ότι όντως υπάρχει ένας τέτοιος λογαριασμός, δηλαδή ένας λογαριασμός με τα στοιχεία που μόλις πληκτρολογήσατε. Πώς το κάνει αυτό; Το Facebook πρέπει ν' ανοίξει τη βάση δεδομένων με τους χρήστες του και να ψάξει σ' αυτή. Όταν λέμε "βάση δεδομένων", σκεφτείτε κάτι που μοιάζει με τον τηλεφωνικό κατάλογο. Μόνο που αντί για ονόματα έχει usernames, κι αντί για τηλέφωνα έχει passwords. Η δουλειά του Facebook είναι να εντοπίσει, μέσα σ' αυτό το "βιβλίο", το username και το password σας. Ας υποθέσουμε λοιπόν ότι το username σας είναι το obi-wan (https://en.wikipedia.org/wiki/Obi-Wan_Kenobi). Τίθεται το ερώτημα: Θα ξεκινήσει το Facebook να ψάχνει από το "A" ή θα αρχίσει από κάπου στη μέση;

Ό,τι συζητάμε ως τώρα, είναι το κλασικό πρόβλημα της αναζήτησης (search problem). Υπάρχουν διάφοροι αλγόριθμοι που δίνουν λύση στο πρόβλημα, ο καθένας με το δικό του τρόπο. Ο αλγόριθμος στον οποίο επικεντρωνόμαστε περιγράφει τη λεγόμενη Δυαδική Αναζήτηση ή αλλιώς το Binary Search.

Ο αλγόριθμος του binary search επιδρά πάντα επί μιας ταξινομημένης λίστας στοιχείων. Ή, αλλιώς, λέμε ότι ο αλγόριθμος δέχεται στην είσοδό του μια ταξινομημένη λίστα. Ένα λεξικό, για παράδειγμα, είναι ταξινομημένο αλφαβητικά ως προς το επίθετο, επομένως επί του λεξικού είναι δυνατό να χρησιμοποιήσουμε τον αλγόριθμο του binary search. Αν το στοιχείο που αναζητάμε υπάρχει στη λίστα, τότε ο αλγόριθμος επιστρέφει τη *θέση* του στοιχείου. Αν πάλι το στοιχείο που αναζητάμε δεν υπάρχει στη λίστα, ο αλγόριθμος δεν επιστρέφει κάτι (ή επιστρέφει το "τίποτα", NULL).

Για παράδειγμα, έστω ότι ψάχνουμε στον τηλεφωνικό κατάλογο την εταιρεία Parabing Creations. Αν επιστρατεύαμε τον binary search κι εκείνος την έβρισκε, θα μας έλεγε ότι το υπό αναζήτηση στοιχείο (Parabing Creations) βρίσκεται στη θέση τάδε (π.χ., 4424) του τηλεφωνικού καταλόγου. Αν η Parabing Creations δεν υπήρχε στον κατάλογο, τότε ο αλγόριθμος θα μας επέστρεφε το NULL (δηλαδή το "τίποτα").

Πώς λειτουργεί η δυαδική αναζήτηση;

Αφού είδαμε το πρόβλημα που λύνει ο αλγόριθμος της δυαδικής αναζήτησης, τώρα ήρθε η στιγμή να δούμε και τη λογική πίσω από τη λύση. Αν φαντάζεστε ότι χρειάζονται τρομακτικά μαθηματικά, σίγουρα θα εκπλαγείτε με την απλότητα του όλου ζητήματος. Ας χρησιμοποιήσουμε ένα συγκεκριμένο παράδειγμα, έτσι, για να κατα-

λάβουμε καλύτερα. Βάζουμε ένα νούμερο στο μυαλό μας, από το 1 μέχρι το 100. Τα επόμενα κουτάκια έχουν όλα τα νούμερα, από το 1 έως το 100, ταξινομημένα κατ' αύξουσα σειρά:

1	2	3	4	5	.....	96	97	98	99	100

Ζητάμε τώρα από τον αναγνώστη να βρει τον αριθμό που βάλαμε στο μυαλό μας. Το ζητάμε αυτό από κάθε αναγνώστη, αλλά θα κερδίσει εκείνος που θα βρει τον αριθμό με τις λιγότερες δυνατές προσπάθειες. Μπορείτε λοιπόν να ρωτάτε κι εμείς από τη μεριά μας θα σας λέμε αν ο αριθμός που μαντέψατε είναι μικρότερος ή μεγαλύτερος ή ακριβώς εκείνος που έχουμε κατά νου. Ας υποθέσουμε ότι κάποιος αναγνώστης αρχίζει και μας ρωτά για όλους τους αριθμούς με τη σειρά, ξεκινώντας από το 1.

Αναγνώστης: Είναι το 1;

deltaHacker: Όχι, το 1 είναι μικρότερο από τον αριθμό μας.

~~1~~	2	3	4	5	.....	96	97	98	99	100

Αναγνώστης: ΟΚ. Είναι μήπως το 2;

deltaHacker: Όχι, το 2 είναι μικρότερο από τον αριθμό μας.

~~1~~	~~2~~	3	4	5	.....	96	97	98	99	100

.....

Αναγνώστης: Βρε 'σείς, το 96 είναι;!

deltaHacker: Μπα που να μη σώναμε, όχι, το 96 είναι μικρότερο από τον αριθμό μας.

~~1~~	~~2~~	~~3~~	~~4~~	~~5~~	.....	~~96~~	97	98	99	100

Τα παραπάνω λοιπόν αποτελούν μια πολύ συγκεκριμένη προσέγγιση στη λύση του προβλήματος της αναζήτησης. Θα συμφωνήσετε, φανταζόμαστε, ότι αυτή η σειραϊκή μέθοδος είναι εξαιρετικά αργή. Και είναι εξαιρετικά αργή γιατί μετά από κάθε λάθος προσπάθεια, ένα και μόνο ένα στοιχείο αφαιρείται από τη λίστα. Αν λοιπόν η λίστα έχει k το πλήθος στοιχεία, τότε στη *χειρότερη* περίπτωση χρειάζονται k το πλήθος προσπάθειες έως ότου εντοπίσουμε το επιθυμητό στοιχείο (ή έως ότου βεβαιωθούμε ότι δεν υπάρχει στη λίστα).

Ας δούμε μια καλύτερη προσέγγιση. Θυμηθείτε τι κάναμε στα παραδείγματα με τον τηλεφωνικό κατάλογο και το λεξικό: την αναζήτηση δεν την ξεκινούσαμε από την αρχή ή από το τέλος, αλλά από τη μέση. Σκεφτόμαστε λοιπόν έναν άλλον αριθμό μεταξύ 1 και 100, σας ζητάμε να τον μαντέψετε, εσείς δεχόσαστε. Μόνο που τώρα δεν εφαρμόζετε τη σειραϊκή μέθοδο, αλλά μία άλλη που παραπέμπει στο "διαίρει και βασίλευε".

Αναγνώστης: Είναι το 50;

deltaHacker: (Ώπα!) Εχμ, όχι, μικρό είναι το 50.

~~1~~	~~2~~	~~3~~	~~4~~	...	~~50~~	...	97	98	99	100

Είναι εντυπωσιακό, αλλά με μία μόνο αποτυχημένη προσπάθεια πετάξατε τη μισή λίστα. Ξεχνάτε λοιπόν τα στοιχεία μεταξύ 1 και 50 κι εστιάζετε την προσοχή σας στη λίστα με τα στοιχεία από το 51 έως το 100 – ή τουλάχιστον αυτό θα κάναμε εμείς, αν ήμασταν στη θέση σας.

Αναγνώστης: Μήπως είναι το 75;

deltaHacker: (Νταρν ιτ, χι μαστ μπι όντου σάμθινγκ.) Ε, όχι, όχι, μεγάλο είναι το 75.

Και πάλι δεν μαντέψατε τον αριθμό, αλλά τουλάχιστον ξεμπερδέψατε με το (δεύτερο) μισό του (δεύτερου) μισού που απέμεινε από την πρώτη προσπάθεια. Η επόμενη στάση σας είναι περισσότερο από αναμενόμενη. Η μέση μεταξύ 51 και 75 είναι το 63:

Αναγνώστης: Ελάτε, πείτε, το 63 είναι;

deltaHacker: Όχι, μεγάλο είναι το 63...

Έχει μείνει η λίστα με τα στοιχεία από το 51 έως και το 62, η μέση είναι το 57, οπότε ρωτάτε:

Αναγνώστης: Το 57 είναι;

deltaHacker: Ναι μωρέ, το 57 είναι, σπουδαίο κατόρθωμα – μπιγκ γουπ!

Κι αυτό, κυρίες και κύριοι, ήταν ο αλγόριθμος της δυαδικής αναζήτησης. Ναι, μόλις γνωρίσατε τον πρώτο σας αλγόριθμο, ο οποίος μάλιστα είναι και ιδιαίτερα αποδοτικός. Νομίζουμε ότι έχει ενδιαφέρον να δούμε πόσους αριθμούς αποκλείουμε σε κάθε βήμα μέχρι να μαντέψουμε τον αριθμό, λαμβάνοντας υπόψη τη χειρότερη περίπτωση. Στο πλαίσιο του παραδείγματός μας, μιλώντας για "χειρότερη περίπτωση" εννοούμε ότι βρίσκουμε τον αριθμό κάνοντας το *μέγιστο* δυνατό αριθμό προσπαθειών.

Πλήθος στοιχείων: 100

Προσπάθεια 1: αποκλείουμε 50 στοιχεία

Προσπάθεια 2: αποκλείουμε 25 στοιχεία

Προσπάθεια 3: αποκλείουμε 13 στοιχεία

Προσπάθεια 4: αποκλείουμε 7 στοιχεία

Προσπάθεια 5: αποκλείουμε 4 στοιχεία

Προσπάθεια 6: αποκλείουμε 2 στοιχεία

Προσπάθεια 7: αποκλείουμε 1 στοιχείο

Καταλαβαίνουμε, λοιπόν, ότι οποιονδήποτε αριθμό μεταξύ 1 και 100 προσπαθήσουμε να μαντέψουμε, θα τον βρούμε με επτά *το πολύ* προσπάθειες. Ας υποθέσουμε τώρα ότι ψάχνουμε τη θέση μιας συγκεκριμένης λέξης, μέσα σε ένα λεξικό με 200.000 λέξεις. Βάζοντας λοιπόν μια λέξη κατά νου, πόσες προσπάθειες θα χρειαστούμε στο *χειρότερο δυνατό* σενάριο πριν βρούμε τη θέση της στο λεξικό; Δείτε:

Πλήθος στοιχείων: 200.000

Προσπάθεια 1: αποκλείουμε 100.000 στοιχεία

Προσπάθεια 2: αποκλείουμε 50.000 στοιχεία

Προσπάθεια 3: αποκλείουμε 25.000 στοιχεία

Προσπάθεια 4: αποκλείουμε 12.500 στοιχεία

Προσπάθεια 5: αποκλείουμε 6.250 στοιχεία

...

Προσπάθεια 14: αποκλείουμε 13 στοιχεία

Προσπάθεια 15: αποκλείουμε 7 στοιχεία

Προσπάθεια 16: αποκλείουμε 4 στοιχεία

Προσπάθεια 17: αποκλείουμε 2 στοιχεία

Προσπάθεια 18: αποκλείουμε 1 στοιχείο

(Αν σε μία δεδομένη στιγμή έχουμε k το πλήθος στοιχεία, στην επόμενη προσπάθεια μένουν [k/2] το πλήθος στοιχεία. Με τα άγκιστρα εννοούμε ότι αν το αποτέλεσμα της διαίρεσης k/2 είναι δεκαδικός αριθμός, τότε παίρνουμε τον πλησιέστερο μεγαλύτερο ακέραιο. Παράδειγμα: Επειδή 13/2 = 6,5 (δεκαδικός αριθμός), ισχύει [13/2] = 7.

Χρησιμοποιώντας τη μέθοδο του binary search, λοιπόν, ψάχνοντας μια δεδομένη λέξη μέσα σε ένα λεξικό 200.000 λέξεων, βρίσκουμε τη θέση της –ή διαπιστώνουμε ότι δεν υπάρχει στο λεξικό–, με 18 προσπάθειες στη χειρότερη περίπτωση. Μιλάμε για τρομακτική βελτίωση σε σχέση με τη σειραϊκή αναζήτηση, η οποία στη χειρότερη περίπτωση θα έπαιρνε 200.000 προσπάθειες.

Γενικά, σε κάθε λίστα με k στο πλήθος στοιχεία, η δυαδική αναζήτηση χρειάζεται log2k το πλήθος βήματα πριν δώσει αποτέλεσμα – και πάντα στη χειρότερη περίπτωση.

Λογάριθμοι

Ως μαθητές σχεδόν πάντα ξεχνούσαμε την έννοια του λογαρίθμου, αλλά σχεδόν πάντα θυμόμασταν την έννοια των δυνάμεων. Όταν λοιπόν κάποιος σας ζητά να του πείτε το αποτέλεσμα του $\log_{10}100$, αυτό που πραγματικά σας ρωτάει είναι: πόσα 10άρια πρέπει να πολλαπλασιάσω μεταξύ τους, έτσι ώστε όλα μαζί να δώσουν 100; Η απάντηση είναι 2, διότι 10 x 10 = 100. Ισχύει, λοιπόν, $\log_{10}100 = 2$. Οι λογάριθμοι και οι δυνάμεις, είναι έννοιες που συνδέονται στενά. Γενικά, ισχύει:

$x^y = z$ όταν και μόνον όταν $\log_x z = y$

Φαίνεται παράξενο ή δυσνόητο; Κάντε μερικές δοκιμαστικές αναθέσεις για τα x και y (το z είναι το αποτέλσμα του x εις την y) και δείτε μόνοι σας. Παράδειγμα: για x = 2 και y = 3, έχουμε $2^3 = 8$ διότι $\log_2 8 = 3$. Επίσης: για x = 2 και y = 4, έχουμε $2^4 = 16$ διότι $\log_2 16 = 4$.

Να σημειώσουμε εδώ ότι, μελετώντας τη βιβλιογραφία, θα διαπιστώνετε ότι αντί για $\log_2 x$ συχνά γράφεται logx. Η βάση 2, μ' άλλα λόγια, συχνά υπονοείται και συνεπώς παραλείπεται. Διαφορετικά: Όταν κάποιος συγγραφέας γράφει logx, είναι πιθανό να εννοεί το δυαδικό λογάριθμο του x. (Σ.τ.Ε. Στη Μαθηματική και υπόλοιπη βιβλιογραφία, βέβαια, είθισται με logx να συμβολίζουμε το δεκαδικό λογάριθμο $\log_{10} x$. Η διευκρίνιση κρίνεται απαραίτητη, ώστε να μη σας δημιουργήσουμε άθελά μας σύγχυση ως προς τους διαφορετικούς συμβολισμούς.)

Για το τέλος αυτής της σημείωσης, σας έχουμε ένα ερώτημα – ή μάλλον πρόκληση: Μπορείτε ν' αποδείξετε *διαισθητικά* γιατί η δυαδική αναζήτηση μέσα σε μια λίστα k στοιχείων χρειάζεται $\log_2 k$ προσπάθειες στη χειρότερη περίπτωση; Τα όποια σχόλιά σας μπορείτε να τ' αφήσετε στο site του περιοδικού, κάτω από τη σχετική δημοσίευση.

Ώρα για κώδικα

Κάτι που δεν πρέπει να ξεχνάμε με τον αλγόριθμο της δυαδικής αναζήτησης, είναι ότι έχει νόημα μόνον όταν εφαρμόζεται σε μια *ταξινομημένη* λίστα. Είναι πιστεύουμε φανερό γιατί πρέπει να ισχύει κάτι τέτοιο, το αναφέρουμε ωστόσο διότι κάποιοι, κάποιες φορές, τείνουν να το ξεχνούν. Αλλά ας αφήσουμε τι κάνουν κάποιοι (κάποιες φορές) κι ας δούμε ένα παράδειγμα προγραμματιστικής υλοποίησης του αλγορίθμου με τη βοήθεια της Python.

Για τη συγγραφή του κώδικα κάνουμε χρήση των arrays. Όσοι δεν είστε γνώστες της προαναφερθείσας δομής, μην ανησυχείτε. Αρκεί να γνωρίζετε ότι μπορείτε να αποθηκεύσετε μία ακολουθία στοιχείων σε μία σειρά από κουτάκια, την οποία σειρά ονομάζουμε array ή αλλιώς πίνακα. Η αρίθμηση των κουτιών του πίνακα ξεκινά από το μηδέν (0). Βλέπετε, σε αντίθεση με τους φυσιολογικούς ανθρώπους, οι προγραμματιστές ξεκινούν να μετράνε πάντα από το 0 κι όχι από το 1. Οπότε το πρώτο στοιχείο του πίνακα θα βρίσκεται στην θέση (κουτάκι) 0, το δεύτερο στοιχείο στην θέση 1, το τρίτο στοιχείο στη θέση 2 και γενικά το n-οστό στοιχείο στη θέση (n-1).

Ως δεδομένα εισόδου, ο αλγόριθμος binary search χρειάζεται δύο πράγματα από εμάς προκειμένου να λειτουργήσει:

1. μια ταξινομημένη λίστα στοιχείων
2. ένα στοιχείο, τη θέση του οποίου θα αναζητήσει εντός της ταξινομημένης λίστας.

Το αποτέλεσμα του αλγορίθμου, δηλαδή η έξοδός του, είναι είτε η θέση του υπό αναζήτηση στοιχείου στη λίστα (αν το στοιχείο υπάρχει) είτε το None (αν το στοιχείο δεν υπάρχει). Ο κώδικας που παραθέτουμε υλοποιεί τον αλγόριθμο της δυαδικής αναζήτησης.

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
```

```
def binary_search(list, guess):
  left = 0                    # αριστερό όριο της λίστας
  right = len(list) - 1       # δεξιό όριο της λίστας
  while left <= right:        # όσο υπάρχει τουλάχιστον ένα στοιχείο που δεν
                                έχουμε ελέγξει...
    mid = (left + right) / 2  # υπολογισμός της μέσης της (υπο)λίστας
    item = list[mid]          # διάβασμα του στοιχείου στη μέση της λίστας
    if item == guess:         # είναι ίσο με αυτό που αναζητάμε...
      return mid              # ...άρα επιστρέφουμε τη θέση του
    if item > guess:          # είναι μεγαλυτερο από αυτό που αναζητάμε...
      right = mid - 1         # ...άρα μετατοπίζουμε το δεξιό άκρο
    else:                     # είναι μικρότερο από αυτό που αναζητάμε...
      left = mid + 1          # ...άρα μετατοπίζουμε το αριστερό άκρο
  return None           # ό,τι αναζητάμε δεν υπάρχει στη λίστα :/
theList = [1, 2, 5, 7, 11, 129, 49250]
print binary_search(theList, 5)
print binary_search(theList, 314)
```

Νομίζουμε ότι ο κώδικας δεν χρειάζεται επεξήγηση: ένα από τα καλά της Python είναι ότι "διαβάζεται" εύκολα, ακόμη κι από κάποιον που πρώτη φορά καταπιάνεται μαζί της. Υπάρχουν άλλωστε και τα σχόλια, τα οποία επίσης βοηθούν. Σας προτείνουμε να τον πληκτρολογήσετε και να τρέξετε το αντίστοιχο προγραμματάκι. Κάντε αν είναι κι αλλαγές, ειδικά στις τρεις τελευταίες γραμμές. Συγκεκριμένα, στην 3η από το τέλος γραμμή ορίζουμε τη λίστα επί της οποίας γίνονται οι αναζητήσεις. Στην προτελευταία γραμμή κάνουμε μια δοκιμαστική αναζήτηση για το στοιχείο 5. Το αποτέλεσμα θα είναι η θέση 2 (όπως τη βλέπει η Python) ή διαφορετικά η 3 (όπως τη μετράμε ξεκινώντας από τ' αριστερά της λίστας). Στην τελευταία γραμμή κάνουμε άλλη μια δοκιμαστική αναζήτηση, αυτή τη φορά για το στοιχείο 314. Τώρα το αποτέλεσμα είναι None, αφού το 314 δεν υπάρχει στη λίστα. Και μια τελευταία παρατήρηση: Κάτω από το while, εκεί που υπολογίζεται το μέσο της λίστας με τη διαίρεση (left + right) / 2, έχετε υπόψη ότι, αν χρειάζεται, η Python στρογγυλοποιεί το αποτέλεσμα στον πλησιέστερο ακέραιο από πάνω.

Ο χρόνος τρέχει

Για ένα δεδομένο πρόβλημα το πιθανότερο είναι να υπάρχουν περισσότερες από μία γνωστές λύσεις, μ' άλλα λόγια περισσότεροι από ένας γνωστοί αλγόριθμοι. Η επιλογή του κατάλληλου αλγορίθμου δεν είναι πάντα εύκολη υπόθεση, αφού εξαρτάται από τη φύση του προβλήματος, τα δεδομένα εισόδου, τις απαιτήσεις σε χώρο και χρόνο κ.ο.κ. Ένα από τα πλέον σημαντικά κριτήρια για την επιλογή αλγορίθμου είναι ο χρόνος εκτέλεσης (running time).

Ας πάρουμε όμως τα πράγματα από την αρχή. Όπως είδαμε στο παράδειγμα της σειραϊκής αναζήτησης, ελέγχοντας εξαντλητικά ένα προς ένα τα στοιχεία μιας λίστας, για 100 στοιχεία χρειαζόμαστε το πολύ 100 προσπάθειες. Ομοίως, για 10 δισεκατομμύρια στοιχεία χρειαζόμαστε το πολύ 10 δισεκατομμύρια προσπάθειες κ.ο.κ. Σε κάθε περίπτωση, είναι προφανές ότι μιλάμε για τη χειρότερη περίπτωση: το στοιχείο που ψάχνουμε είτε είναι στο τέλος της λίστας, είτε δεν υπάρχει καν στη λίστα. Η σειραϊκή αναζήτηση, λένε Μαθηματικοί κι Επιστήμονες της Πληροφορικής, αποτελεί ένα παράδειγμα αλγορίθμου γραμμικού χρόνου (linear time algorithm). Προφανώς, το κακό με τους αλγορίθμους γραμμικού χρόνου είναι ότι διαρκούν όλο και πιο πολύ, όσο το πλήθος των στοιχείων που πρέπει να ελέγχουν αυξάνει. Αν σας φαίνεται κακή η απόδοσή τους, σας λέμε ότι έχετε δίκιο: είναι πράγματι κακή. Υπάρχουν όμως και χειρότερα, όπως, π.χ., αλγόριθμοι τετραγωνικού χρόνου, όπου ο χρόνος ολοκλήρωσης για τη χειρότερη περίπτωση είναι ανάλογος του τετραγώνου του πλήθους των δεδομένων εισόδου. Αλλά ας μην προτρέχουμε.

Τα πράγματα είναι εντελώς διαφορετικά με τον αλγόριθμο του binary search. Είδαμε πως για μία λίστα 100 στοιχείων, χρειαζόμαστε 7 το πολύ προσπάθειες (τουλάχιστον 14 φορές πιο γρήγορος από τη σειραϊκή αναζήτηση, στη χειρότερη περίπτωση). Αν είχαμε 10 δισεκατομμύρια στοιχεία, θα χρειαζόμασταν 33 το πολύ προσπάθειες (τουλάχιστον 303 εκατομμύρια φορές πιο γρήγορος από τη σειραϊκή αναζήτηση, στη χειρότερη περίπτωση). Καταλαβαίνετε για τι διαφορά μιλάμε, έτσι δεν είναι; Σίγουρα το καταλαβαίνετε – και μόλις τώρα σας εξηγήσαμε την έννοια του λογαριθμικού χρόνου (logarithmic time).

Ας βάλουμε τα στοιχεία κάτω, σε έναν πίνακα.

Σειραϊκή Αναζήτηση	**Δυαδική Αναζήτηση**
100 στοιχεία 100 προσπάθειες για τη χειρότερη περίπτωση	100 στοιχεία 7 προσπάθειες για τη χειρότερη περίπτωση
10.000.000.000 στοιχεία 10.000.000.000 προσπάθειες για τη χειρότερη περίπτωση	10.000.000.000 33 προσπάθειες για τη χειρότερη περίπτωση
Γενικά: n στο πλήθος στοιχεία n στο πλήθος προσπάθειες για τη χειρότερη περίπτωση	Γενικά: n στο πλήθος στοιχεία $\log_2 n$ στο πλήθος προσπάθειες για τη χειρότερη περίπτωση
γραμμικός χρόνος, **O(n)**	λογαριθμικός χρόνος, **$O(\log_2 n)$**

Μπιγκ Όου!

Ο συμβολισμός big O, τον οποίο σιωπηρά εισαγάγαμε στην τελευταία γραμμή του προηγούμενου πίνακα, χρησιμοποιείται κατά κόρον στο πλαίσιο της βιβλιογραφίας των αλγορίθμων κι εκφράζει ένα από τα πιο σημαντικά πράγματα που απασχολούν

–ή θα τουλάχιστον θα έπρεπε ν' απασχολούν– κάθε προγραμματιστή: την ταχύτητα ενός αλγορίθμου. Αν λοιπόν πρόκειται να χρησιμοποιήσετε τον αλγόριθμο κάποιου άλλου προγραμματιστή, χρήσιμο θα ήταν να γνωρίζετε την ταχύτητά του. Έτσι, θα είστε σε θέση να τον συγκρίνετε με άλλους, γνωστούς ή άγνωστους αλγορίθμους και τελικά να επιλέξετε τον γρηγορότερο.

Ας δούμε λόγο το ακόλουθο σενάριο. Ο κος Ντεβελοπερόπουλος είναι ένας νέος, φιλόδοξος, προγραμματιστής, ο οποίος πρόσφατα κατάφερε και βρήκε δουλειά στην Ευρωπαϊκή Υπηρεσία Διαστήματος (E.S.A.). Από τα πρώτα του καθήκοντα είναι να γράψει ένα πρόγραμμα αναζήτησης, το οποίο θα χρησιμοποιηθεί στην επόμενη αποστολή στο διάστημα. Συγκεκριμένα, όποτε χρειάζεται το πρόγραμμα πρέπει να τρέχει αμέσως και μάλιστα να βρίσκει το υπό αναζήτηση στοιχείο σε 10 το πολύ δευτερόλεπτα. Το στοιχείο αυτό μπορεί να είναι γεωγραφικές συντεταγμένες προσεδάφισης ή οτιδήποτε άλλο. (Οι λεπτομέρειες δεν μας ενδιαφέρουν, αφού δεν πρόκειται να πιάσουμε σύντομα δουλειά στην E.S.A.)

Ο κος Ντεβελοπερόπουλος αρχίζει να σκέφτεται για τον αλγόριθμο που θα πρέπει να υλοποιήσει. Προσπαθεί ν' αποφασίσει μεταξύ της σειραϊκής αναζήτησης και της δυαδικής. Γνωρίζει, προφανώς, ότι η δυαδική αναζήτηση είναι πολύ πιο γρήγορη, ωστόσο ο κώδικας για την υλοποίηση θα είναι αρκετά πιο πολύπλοκος και πιθανώς να έχει bugs. Ο κος Ντεβελοπερόπουλος μόλις έπιασε δουλειά, οπότε το τελευταίο πράγμα που θα ήθελε στον κώδικά του είναι τα bugs. Σκέφτεται, λοιπόν, ότι ίσως πρέπει να υλοποιήσει μια απλή σειραϊκή αναζήτηση, αφού σ' αυτή την περίπτωση η πιθανότητα για bugs είναι σαφώς μικρότερη.

Ο φίλος μας αποφασίζει να χρονομετρήσει, κατά κάποιον τρόπο, τους δύο αλγορίθμους. Γνωρίζει ότι το hardware που θα φιλοξενήσει τον κώδικά του χρειάζεται ένα χιλιοστό του δευτερολέπτου (1ms) για κάθε προσπάθεια. Για 100 στοιχεία, λοιπόν, με τη σειραϊκή αναζήτηση θα χρειαστούν 100ms και με τη δυαδική 7ms. Εντάξει, η δυαδική είναι πολύ πιο γρήγορη από τη σειραϊκή –για την ακρίβεια είναι 100/7 ή περίπου 15 φορές πιο γρήγορη–, όμως σε κάθε περίπτωση το πρόγραμμα επιστρέφει απάντηση εντός του επιτρεπτού χρονικού διαστήματος (10sec).

"Εδώ που τα λέμε βέβαια", σκέφτεται ο κος Ντεβελοπερόπουλος, "σιγά μην έχω λίστα 100 στοιχείων. Κοτζάμ ι-ες-έι είν' αυτή, οπότε για να 'μαι μέσα ας λάβω υπόψη λίστα με ένα δισεκατομμύρια στοιχεία". Αυτή τη φορά –κι επειδή τον γοητεύουν κάπως οι λογάριθμοι– υπολογίζει απευθείας το δυαδικό λογάριθμο του 1.000.000.000 και τον βρίσκει περίπου ίσο προς 32. Ώστε για ένα δις στοιχεία, το binary search χρειάζεται 32ms και η σειραϊκή αναζήτηση, που όπως υπολόγισε προηγουμένως είναι 15 φορές πιο αργή, θα χρειάζεται 32 x 15 = 480ms. "Μάλιστα. Ωραία. Ακόμη και με τόσα πολλά στοιχεία πάλι μένω εντός του χρονικού ορίου, άσχετα αν χρησιμοποιήσω binary search ή απλή, σειραϊκή αναζήτηση".

Ο κος Ντεβελοπερόπουλος, που κατά τη διάρκεια των σπουδών του δεν πήγαινε στις παραδόσεις και δεν πολυδιάβαζε, μόλις έκανε ένα τεράστιο λάθος. Αγαπούσε και αγαπάει τον προγραμματισμό, όμως θα έπρεπε να είχε μάθει να σκέφτεται περισσότερο συστηματικά. Δείτε στον ακόλουθο πίνακα το μέγεθος της επιπολαιότητάς του.

	Σειραϊκή Αναζήτηση	**Δυαδική Αναζήτηση**
100 στοιχεία	100 ms	7 ms
1.000.000.000 στοιχεία	11,5 μέρες	32 ms

Για 1 δις στοιχεία η σειραϊκή αναζήτηση χρειάζεται 1 δις χιλιοστά του δευτερολέπτου, δηλαδή 1 εκατομμύριο δευτερόλεπτα ή αλλιώς εντεκάμιση περίπου μέρες! Γιατί όμως έκανε αυτό το λάθος ο φίλος μας; Πολύ απλά, διότι ο ρυθμός μεταβολής (επιτάχυνση) του χρόνου ολοκλήρωσης που παρατηρείται σε καθέναν από τους δύο αλγορίθμους, είναι διαφορετικός. Το λέμε κι αλλιώς: Ο γραμμικός χρόνος της σειραϊκής αναζήτησης αυξάνει γρηγορότερα σε σχέση με το λογαριθμικό χρόνο της δυαδικής αναζήτησης. Ακριβώς γι' αυτό *δεν* δικαιούμαστε να βρούμε το λόγο τον χρόνων, π.χ., για 100 στοιχεία, και τον ίδιο λόγο να χρησιμοποιήσουμε και για τα 1 δις στοιχεία.

Το προηγούμενο παράδειγμα φανερώνει ότι δεν αρκεί να γνωρίζουμε το χρόνο που χρειάζεται ένας αλγόριθμος για είσοδο n στο πλήθος στοιχείων, αλλά πρέπει επίσης να γνωρίζουμε και πώς ο χρόνος αυτός αλλάζει καθώς το n αυξάνει. Εδώ λοιπόν έρχεται ο συμβολισμός big O και μας δίνει ακριβώς αυτή την πληροφορία.

Γράφοντας, π.χ., ότι ένας αλγόριθμος είναι O(n), εννοούμε ότι στη χειρότερη περίπτωση χρειάζεται χρόνο ανάλογο του πλήθους των στοιχείων επί των οποίων εφαρμόζεται. Άλλο παράδειγμα: Αν ένας αλγόριθμος είναι $O(n^2)$, τότε στη χειρότερη περίπτωση χρειάζεται χρόνο ανάλογο του *τετραγώνου* του πλήθους των στοιχείων επί των οποίων εφαρμόζεται. Από μια απλή γραφική παράσταση των συναρτήσεων f(x)=x και $g(x)=x^2$ (χρησιμοποιήστε το WolframAlpha), βλέπουμε ότι η g μεγαλώνει πολύ πιο γρήγορα σε σχέση με την f (για x > 1). Έτσι, καταλαβαίνουμε ότι ένας αλγόριθμος $O(n^2)$ είναι πιο αργός σε σχέση με έναν αλγόριθμο O(n).

Όχι μόνο το χειρότερο σενάριο

Ο συμβολισμός big O εκφράζει το χειρότερο δυνατό σενάριο. Προφανώς αυτό είναι κάτι που μας ενδιαφέρει αρκετά, καθώς μπορούμε να ξέρουμε με ασφάλεια το μέγιστο χρόνο που θα κάνει το πρόγραμμά μας να βρει τη λύση στο πρόβλημα που αντιμετωπίζει. Υπάρχουν όμως και περιπτώσεις όπου εκτός από το χειρότερο σενάριο μάς ενδιαφέρει και η συμπεριφορά του αλγορίθμου κατά τη μέση περίπτωση. Αλλά αυτή η συζήτηση αποτελεί το αντικείμενο επόμενου άρθρου.

Προς το παρόν, ελπίζουμε να σας άρεσε αυτή η πρώτη βόλτα που κάναμε στη χώρα των αλγορίθμων. Αν σας άνοιξε η όρεξη και τώρα θέλετε μεγαλύτερη βόλτα, να ξέρετε πως θα ήταν τιμή μας αν μας επιτρέπατε να σας ξεναγήσουμε.

Καλή σας διασκέδαση!

Skill: Beginner
Tag: version control, revision control, source control, VCS, Git

VCS και Git: Τι στο καλό είναι και γιατί πρέπει να νοιαστείτε

Στον κόσμο των Version Control Systems το Git έχει φέρει τα πάνω κάτω, ωστόσο επτά περίπου από εσάς δεν έχετε ιδέα για το τι γράψαμε μόλις. Αλλά μην ανησυχείτε. Η παρούσα μίνι σειρά άρθρων ετοιμάστηκε για εσάς, τους επτά. Οι υπόλοιποι μπορείτε να το αγνοήσετε με ασφάλεια. Διαβάστε αν είναι κάποιο άλλο από τα άρθρα του τεύχους ή ένα άρθρο από άλλο τεύχος. Εδώ δεν έχει κάτι για εσάς. Αλήθεια σας λέμε.

του Χρήστου Βαρελά

Η διαχείριση των τροποποιήσεων σε έγγραφα, δικτυακούς τόπους, προγράμματα κ.λπ. αναφέρεται ως version control (έλεγχος εκδόσεων) ή revision control (έλεγχος αναθεωρήσεων) ή αλλιώς source control (έλεγχος πηγαίου κώδικα). Παραδοσιακά, οι αλλαγές αντικατοπτρίζονται από έναν αριθμό ή/και ένα γράμμα και σε κάθε μία από αυτές αντιστοιχίζεται ένα timestamp.

Για παράδειγμα, ένας υπάλληλος εταιρείας που προετοιμάζει μια αναφορά προόδου για το τρίμηνο που πέρασε, ξεκινά μ' ένα έγγραφο το οποίο αρχικά ονομάζει report-r1.doc. Μετά από μία εβδομάδα, αφού έχει συγκεντρώσει αρκετά στοιχεία από συνεργάτες, ίσως έχει φτάσει στο report-r4.doc. Επειδή δεν είναι σίγουρος για το πόσο κοντά βρίσκεται στην τελική αναφορά (ίσως αφήσει εκτός συγκεκριμένους πίνακες ή γραφήματα), έχει κρατήσει και τα έγγραφα report-r2.doc και report-r3.doc. Κάποια στιγμή φτάνει στο report-r5.doc και του αρέσει. Πριν όμως στείλει την αναφορά στη Διεύθυνση, σκέφτεται ότι καλό είναι να τη δει και κάποιος που γνωρίζει καλύτερα γραμματική και συντακτικό. Εντάξει, για το τεχνικό μέρος δεν έχει κάποια αμφιβολία, όμως κρίμα είναι να ρεζιλευτεί με κανένα "ανεξαρτήτου ηλικίας" ή "εκ των ουκ άνευ". Ειδοποιεί λοιπόν την κα Διορθώτρια (TM) από το GrammarNazis department κι αμέσως της στέλνει το report-r5.doc. Όπως αναμενόταν, εκείνη εντοπίζει αρκετά τραγικά λάθη. Τα διορθώνει ξεφυσώντας και του επιστρέφει τη νέα εκδοχή της αναφοράς, ονόματι report-r5d.doc ("d", από το "dior8wmeno": για κάποιο μυστήριο λόγο, στο GrammarNazis department χρησιμοποιούν greeklish στα ονόματα των αρχείων). Ο φίλος μας ρίχνει μια γρήγορη ματιά στο νέο έγγραφο και πραγματικά δεν καταλαβαίνει τι ακριβώς "διόρθωσε" (κάνει air-quotes) η κα Διορθώτρια (TM). Κρατάει ωστόσο το report-r5d.doc –ποτέ δεν ξέρεις– και στο τέλος προσθέτει μια ακόμα

παράγραφο, η οποία πιστεύει ότι θα κάνει τη διαφορά. Το τελικό έγγραφο το ονομάζει report-r5-final.doc, στη Διεύθυνση όμως στέλνει ένα αντίγραφό του που ονομάζει Q1report.doc. Όλα καλά.

Στο προηγούμενο παράδειγμα, τα γράμματα και οι αριθμοί στα ονόματα των αρχείων βοηθούν το συντάκτη της αναφοράς να ολοκληρώσει ομαλά την εργασία του. Ακόμη και μετά από μήνες, αν ρίξει μια ματιά στα αρχεία report-r1, ..., report-r5, report-r5d, report-r5d-final και Q1report, μάλλον θα θυμηθεί εύκολα σε τι αφορούν. Όταν μάλιστα φτάσει η ώρα για άλλη μια αναφορά, ίσως δεν τη ξεκινήσει από την αρχή αλλά από κάποιο εξ αυτών των αρχείων.

Γιατί VCS;

Σε σχετικά μικρά κι απλά πρότζεκτ, όπου έχουμε έναν μόνον εργαζόμενο ή έστω μια μικρή ομάδα ανθρώπων που συνεργάζονται, η διαχείριση των αλλαγών καθ' όλη τη διάρκεια του έργου είναι πράγματι δυνατό να διευκολύνεται μόνο με την κατάλληλη ονομασία ενός συνόλου αρχείων, τα οποία μεταβάλλονται. Η κατάσταση όμως αλλάζει δραματικά όταν μιλάμε για πιο περίπλοκα ή/και πιο μεγάλα πρότζεκτ, μάλιστα ασχέτως αν έχουμε έναν εργαζόμενο ή περισσότερους. Χρειαζόμαστε τότε ένα σύγχρονο Version Control System (VCS) και στη συνέχεια παραθέτουμε μερικούς λόγους που συνηγορούν υπέρ της χρήσης του.

- Αναπτύσσετε μια εφαρμογή *μόνος σας* και ήδη έχετε γράψει αρκετό κώδικα, πιθανώς σε περισσότερα από ένα αρχεία. Ξαφνικά θέλετε να δοκιμάσετε μια νέα ιδέα που μόλις σκεφτήκατε. Θα ήταν βολικό αν μπορούσατε να ξεκινήσετε εύκολα ένα νέο branch, δηλαδή ένα νέο πρότζεκτ με βάση το υπάρχον, στο οποίο θα μπορέσετε με ασφάλεια να δοκιμάσετε τη νέα σας ιδέα. Φυσικά, όταν μιλάμε για νέο πρότζεκτ δεν εννοούμε ότι ξεκινάτε από την αρχή. Αντίθετα, συνεχίζετε από το σημείο στο οποίο ήδη βρίσκεται το πρότζεκτ σας αλλά τώρα έχετε κι ένα δεύτερο πρότζεκτ που εξελίσσεται προς μια νέα κατεύθυνση. Τα αρχεία του παλιού πρότζεκτ μένουν ως έχουν και μάλιστα μπορείτε να δουλεύετε στο πρωτότυπο πρότζεκτ παράλληλα με το νέο – ας το πούμε πειραματικό.
- Σε περίπτωση που η ιδέα σας αποδειχθεί καλή κι επιφέρει καρπούς, θα ήταν ευχής έργο αν μπορούσατε να συγχωνεύσετε (merge) με ακρίβεια τις όποιες αλλαγές στο αρχικό πρότζεκτ.
- Αν πάλι η ιδέα σας αποδειχθεί αδιέξοδη, θα ήταν επίσης ευχής έργο αν επιστρέφατε στο αρχικό πρότζεκτ ξεχνώντας παντελώς το νέο branch. Εντάξει, αυτό γίνεται εύκολα, αρκεί, π.χ., να χρησιμοποιήσετε δύο κατάλληλα ονομασμένους καταλόγους, αλλά χρειάζεται προσοχή διότι πάντα υπάρχει το περιθώριο για σοβαρό λάθος. Για παράδειγμα, δεν είναι καθόλου απίθανο να ξεχάσετε σε ποιον κατάλογο βρίσκεστε και να τροποποιήσετε ή ακόμη και να διαγράψετε τα λάθος αρχεία.

- Ας δούμε τώρα και την περίπτωση κατά την οποία *περισσότεροι από ένας* άνθρωποι δουλεύουν και συνεργάζονται πάνω στο ίδιο πρότζεκτ, κι επιθυμούν να έχουν μια σταθερή βάση αναφοράς (κύριο branch). Φυσικά, καθένας τους θα θέλει να δουλεύει στο δικό του branch κι όποτε χρειάζεται να συγχωνεύει τις αλλαγές στο κύριο.
- Επιπρόσθετα, σκεφτείτε ότι οι συνεργάτες είναι πιθανό να μη βρίσκονται στον ίδιο χώρο εργασίας. Δεν αποκλείεται μάλιστα να έχουν διαφορετικά ωράρια ή ακόμη και να βρίσκονται σε διαφορετικές ζώνες ώρας. Πώς γίνεται να συνεννοούνται όλοι αποτελεσματικά, ώστε η εξέλιξη του πρότζεκτ να προχωρά χωρίς προβλήματα; Χρειάζεται άραγε να συνεννοούνται πριν κάνουν οτιδήποτε ή μήπως το πρότζεκτ θα μπορούσε να προχωρά χωρίς να έρχονται διαρκώς οδηγίες από τα "κεντρικά";
- Ασχέτως της όποιας κεντρικής καθοδήγησης ή της παντελούς απουσίας της, όσοι εργάζονται σε μεγάλα πρότζεκτ είναι καλό να γνωρίζουν –ανά πάσα στιγμή– ποιος συνεργάτης εισήγαγε ποιες αλλαγές στο κύριο branch. Και είναι καλό να το γνωρίζουν χωρίς να επικοινωνούν μεταξύ τους.

Ζητήματα ή και προβλήματα σαν τα προηγούμενα, αντιμετωπίζονται με χρήση του κατάλληλου VCS. Κατ' ελάχιστον, ένα σύστημα ελέγχου εκδόσεων καταγράφει τις αλλαγές που γίνονται σε ένα αρχείο ή σε ένα σύνολο αρχείων, κι επιτρέπει τη μετάβαση από έκδοση σε έκδοση. Αν και τα VCS χρησιμοποιούνται εκτεταμένα σε πρότζεκτ ανάπτυξης λογισμικού, όπου εκεί αυτό που αλλάζει είναι τα αρχεία με πηγαίο κώδικα, είναι κατάλληλα και για άλλους τύπους αρχείων. Έτσι, τα VCS επιστρατεύονται στην πράξη από γραφίστες, συγγραφείς, bloggers, παραγωγούς περιεχομένου κ.ά. Χάρη στα VCS, λοιπόν, είμαστε σε θέση να επαναφέρουμε προηγούμενες εκδόσεις ενός αρχείου ή ολόκληρου του πρότζεκτ, να βλέπουμε τις αλλαγές που συμβαίνουν με το πέρασμα του χρόνου, να γνωρίζουμε ποιος τροποποίησε τι, ποιος εντόπισε πρόβλημα και το ανέφερε κ.ο.κ. Αξίζει να υπογραμμιστεί η δυνατότητα των VCS για ανάκτηση προηγούμενων εκδόσεων: Χάρη σ' αυτή, ακόμη κι αν φέρουμε τα πάνω κάτω και μπερδέψουμε τα πάντα σε απελπιστικό βαθμό, πολύ εύκολα μπορούμε να γυρίσουμε σε μια προηγούμενη, καλή κατάσταση.

Μερικά συστήματα ελέγχου εκδόσεων

Ένα από τα μακροβιότερα VCS είναι το λεγόμενο RCS (Revision Control System, https://www.gnu.org/software/rcs). Κυκλοφόρησε το 1982 και είναι ικανό να παρακολουθεί μεμονωμένα αρχεία αλλά όχι ολόκληρα πρότζεκτ. Ακόμη και σήμερα πάντως το RCS εξακολουθεί να είναι χρήσιμο, ειδικά σε σενάρια όπου έχουμε έναν μόνο χρήστη. Για παράδειγμα, ο διαχειριστής ενός server πιθανώς να καταφύγει στο RCS προκειμένου να διατηρεί διαφορετικές εκδόσεις μεμονωμένων configuration files ή scripts.

Εξέλιξη του RCS αποτελεί το CVS (Concurrent Versioning System, http://savannah.nongnu.org/projects/cvs). Το CVS κυκλοφόρησε πρώτη φορά το 1990, είναι ικανό να χειρίζεται ολόκληρα πρότζεκτ, ακολουθεί τη λογική client-server και υποστηρίζει περισσότερους από έναν χρήστες. Στον κόσμο του CVS, όλα

τα αρχεία ενός πρότζεκτ, μαζί με τις διάφορες εκδόσεις και το ιστορικό της εξέλιξης του έργου, τηρούνται σε έναν κεντρικό server. Οι χρήστες (clients) συνδέονται στον server προκειμένου να πάρουν (check out) στους υπολογιστές τους ένα πλήρες αντίγραφο του πρότζεκτ. Ο καθένας τους εργάζεται πάνω στο τοπικό αντίγραφό του και μετά στέλνει (check in) στον server τις αλλαγές που πραγματοποίησε. Προκειμένου να αποφεύγεται το φαινόμενο των συγκρούσεων (conflicts), ο CVS server δέχεται αλλαγές μόνο για την πλέον πρόσφατη έκδοση ενός οποιουδήποτε αρχείου. Αυτό σημαίνει ότι οι χρήστες ανά πάσα στιγμή οφείλουν να έχουν ενημερωμένο το τοπικό τους αντίγραφο, εφαρμόζοντας τα patches όλων των άλλων χρηστών. Ο CVS client φροντίζει ώστε η ενημέρωση να γίνεται αυτόματα. Όταν όμως προκύπτουν συγκρούσεις μεταξύ μιας checked in αλλαγής και μιας τοπικής αλλαγής, τότε ζητείται η παρέμβαση του χρήστη.

Το δε Apache Subversion (SVN, http://subversion.apache.org), εξάλλου, είναι σχεδιασμένο ως η βελτίωση του CVS και είναι συμβατό μ' αυτό. Κυκλοφόρησε το 2000, χαίρει ευρείας αποδοχής σε κοινότητες ανοικτού κι ελεύθερου λογισμικού και χρησιμοποιείται σε πρότζεκτ όπως το FreeBSD (https://www.freebsd.org), η Free Pascal (http://www.freepascal.org) κι ο GCC (https://gcc.gnu.org).

Απαραίτητη ορολογία

Πριν εστιάσουμε την προσοχή μας στο εξαίρετο Git, αξίζει να αφιερώσουμε λίγο χρόνο συζητώντας για μερικούς όρους που χρησιμοποιούνται με το ίδιο (σχεδόν) νόημα σε κάθε σύστημα ελέγχου εκδόσεων. Οι όροι δεν παρατίθενται αλφαβητικά, αλλά με τέτοια σειρά ώστε κάθε επεξήγηση όρου να μην περιλαμβάνει όρο που ακολουθεί. Έχοντας μια ιδέα για τη σημασία των όρων, όλα όσα παρουσιάζουμε στη συνέχεια κατανοούνται πολύ πιο εύκολα.

Revision. Αναφέρεται στη λεγόμενη αναθεώρηση ή αλλιώς στην έκδοση (**version**) ενός μεμονωμένου αρχείου ή ενός ολόκληρου πρότζεκτ. Κάθε μεταβολή που παγιώνεται, οδηγεί σε μια νέα έκδοση.

Baseline. Πρόκειται για μια εγκεκριμένη έκδοση, από την οποία ενδέχεται ν' ακολουθήσουν κι άλλες εκδόσεις. Το baseline μπορούμε να το φανταζόμαστε ως σημείο ή ως βάση αναφοράς. Κατά περίπτωση, αντί για το baseline χρησιμοποιείται ένας από τους όρους **label** ή **tag**. Ένα baseline/label/tag ορίζει ένα **snapshot**. Σε κάθε πρότζεκτ, ορισμένα snapshots θεωρούνται πιο σημαντικά από άλλα. Αυτά τα ξεχωριστής σημασίας snapshots ενδέχεται να αντιστοιχούν σε δημοσιευμένες εκδόσεις (**published releases**) ή σε ορόσημα (**milestones**).

Branch. Ένα σύνολο αρχείων τα οποία υπόκεινται σε κάποιο VCS, ανά πάσα στιγμή είναι δυνατόν να "διακλαδωθεί" (branched). Αμέσως μετά τη διακλάδωση έχουμε δύο σύνολα αρχείων (baseline και branch) με το καθένα εξ αυτών να ακολουθεί το δικό του μονοπάτι. Από κάθε άποψη, τα δύο αυτά μονοπάτια είναι ανεξάρτητα. Ένα από τα δύο ίσως εξελίσσεται (για την ακρίβεια μεταβάλλεται), ενώ το άλλο ίσως παραμένει στάσιμο. Ίσως πάλι να εξελίσσονται και τα δύο, αλλά το καθένα με διαφορετικό ρυθμό.

Trunk. Έτσι ονομάζεται η μοναδική γραμμή εξέλιξης ενός πρότζεκτ η οποία *δεν* αποτελεί branch. Κατά περιπτώσεις, το trunk (κορμός) ονομάζεται **mainline** ή **master**.

Change. Μία αλλαγή (change ή **diff** ή **delta**) αναπαριστά μια συγκεκριμένη τροποποίηση σε κάποιο αρχείο, το οποίο παρακολουθείται από ένα VCS. Ο βαθμός διακριτικότητας (**granularity**) που πρέπει να έχει μια αλλαγή ώστε να θεωρηθεί ως τέτοια, εξαρτάται από το εκάστοτε VCS.

Repository. Το αποθετήριο (repository ή **depot**) είναι το μέρος όπου βρίσκονται τα αρχεία ενός πρότζεκτ, μαζί με όλα τα μεταδεδομένα (**metadata**) που δείχνουν την έως τώρα εξέλιξή του. Το αποθετήριο συνήθως βρίσκεται σε κάποιον server, ενδέχεται ωστόσο να είναι και τοπικό. Όταν μιλάμε για αρχικοποίηση (**initialization**) εννοούμε τη δημιουργία ενός νέου, κενού αποθετηρίου.

Checkout. Έτσι ονομάζεται η ενέργεια της δημιουργίας ενός τοπικού αντιγράφου των αρχείων ενός αποθετηρίου. Σημειώστε ότι έχουμε δυνατότητα λήψης συγκεκριμένης έκδοσης ή απλά της πλέον πρόσφατης.

Clone. Η κλωνοποίηση ενός αποθετηρίου σημαίνει ότι δημιουργούμε ένα νέο αποθετήριο με όλα τα αρχεία και όλα τα έξτρα δεδομένα που έχει το αρχικό. Έτσι, το κλωνοποιημένο περιλαμβάνει όλες τις αναθεωρήσεις του πρωτότυπου. Δύο αποθετήρια ονομάζονται κλώνοι όταν διατηρούνται συγχρονισμένα, ώστε να περιλαμβάνουν τις ίδιες ακριβώς αναθεωρήσεις.

Commit. Η αντίθετη ενέργεια του checkout είναι το commit (παράδοση) ή αλλιώς **checkin**. Αφού κάνουμε τις αλλαγές μας στο τοπικό αντίγραφο, με τη διαδικασία του commit τις ενσωματώνουμε (**merge**) στο κεντρικό αποθετήριο.

Conflict. Όταν δύο ή περισσότεροι χρήστες επιχειρούν να εισάγουν αλλαγές στο ίδιο αρχείο αλλά το VCS αδυνατεί να τις συμβιβάσει, τότε έχουμε το λεγόμενο conflict (σύγκρουση). Σε τέτοιες περιπτώσεις απαιτείται παρέμβαση από πλευράς κάποιου χρήστη, ο οποίος είτε θα ενσωματώσει όλες τις αλλαγές όπως κρίνει καλύτερα είτε θα επιλέξει μερικές αλλαγές και θ' απορρίψει κάποιες άλλες.

Head. Η λεγόμενη κεφαλή συχνά αναφέρεται κι ως tip (κορυφή) και πάντα "δείχνει" το πλέον πρόσφατο commit, είτε στο trunk είτε σε κάποιο branch. Το trunk και κάθε branch έχουν το δικό τους head, σε ορισμένες περιπτώσεις ωστόσο η κεφαλή του trunk γράφεται με κεφαλαία (HEAD).

Update. Με την ενημέρωση (update) επικαιροποιούμε το τοπικό μας αντίγραφο με όλες τις αλλαγές που έχουν συμβεί στο αποθετήριο από άλλους χρήστες.

Κατανεμημένα VCS

Τα συγκεντρωτικά (centralized) VCS, όπως, π.χ., είναι τα CVS και Subversion, επιτρέπουν σε ομάδες χρηστών να συνεργάζονται ομαλά πάνω στο ίδιο πρότζεκτ. Από τη στιγμή που ο καθένας κάνει τακτικά commits ή/και updates, όλοι είναι σε θέση να γνωρίζουν ποιος εργάζεται πάνω σε τι, ενώ βεβαίως όλοι μπορούν και παρακολουθούν την πορεία του πρότζεκτ. Την ίδια στιγμή, ο διαχειριστής του κεντρικού server ελέγχει τα δικαιώματα κάθε χρήστη κι αποφασίζει για την εξέλιξη του πρότζεκτ.

Τα centralized VCS όμως έχουν και μειονεκτήματα. Ένα από τα σοβαρότερα είναι η απουσία ευελιξίας από μέρους τους, εν όψει προσωρινής βλάβης ή κατάρρευσης. Σκεφτείτε, π.χ., ότι ο κεντρικός server ενός VCS για κάποιο λόγο πέφτει. Μέχρι να είναι και πάλι πλήρως λειτουργικός και διαθέσιμος, κανείς δεν μπορεί να κάνει updates ή commits, ούτε φυσικά να συνεργαστεί με άλλους. Ακόμα χειρότερα, αν ο δίσκος του server χαλάσει και δεν υπάρχει backup, όλο το πρότζεκτ χάνεται και δεν υπάρχει δυνατότητα πλήρους ανακατασκευής του από τα όποια τοπικά αντίγραφα.

Το σοβαρό αυτό μειονέκτημα των συγκεντρωτικών VCS έρχονται ν' αντισταθμίσουν τα κατανεμημένα VCS (Distributed Version Control Systems). Δύο παραδείγματα κατανεμημένων VCS αποτελούν τα Git και Mercurial. Σε καθένα απ' αυτά, οι χρήστες δεν κάνουν checkout μόνο το πλέον πρόσφατο snapshot του πρότζεκτ. Πολύ περισσότερο, κλωνοποιούν ολόκληρο το αποθετήριο. Ακόμη λοιπόν κι αν ο server τιναχτεί στον αέρα με μανία κι όταν προσγειωθεί γίνει κομματάκια και *δεν* υπάρχει ούτε δείγμα backup, ξεκινώντας από έναν οποιονδήποτε πελάτη-κλώνο θα είναι δυνατό να ανακατασκευάσουμε το αποθετήριο.

Η γέννηση του Git

Ένα από τα μεγαλύτερα γνωστά πρότζεκτ Ανοικτού Λογισμικού είναι ο πυρήνας του Linux. Από την πρώτη του έκδοση (0.0.1, ο Linus Torvalds την κυκλοφόρησε στις 17 Σεπτεμβρίου του 1991) έως και το σχετικά πρόσφατο 2002, οι αλλαγές στον πηγαίο κώδικα διακινούνταν υπό τη μορφή patches και archives. Δεν γινόταν χρήση κάποιου VCS, κυρίως λόγω της αποστροφής που έτρεφε ο Linus προς τα συγκεντρωτικά συστήματα του είδους. Το 2002, ωστόσο, η ανάπτυξη του Linux kernel πέρασε υπό την εποπτεία του BitKeeper, ενός κατανεμημένου VCS που ικανοποιούσε τα τεχνικά κριτήρια του Torvalds. Αν και την εποχή το BitKeeper αποτελούσε ιδιόκτητο λογισμικό, στον Linus και σε άλλους μηχανικούς λογισμικού δόθηκε δωρεάν και πλήρης πρόσβαση στο σύστημα.

Τον Απρίλιο του 2005 η BitMover, η εταιρεία πίσω από το BitKeeper, σταμάτησε να υποστηρίζει την ανάπτυξη του Linux kernel και τερμάτισε το ιδιαίτερο καθεστώς δωρεάν χρήσης της υπηρεσίας για τους kernel developers. Αιτία ήταν οι προσπάθειες του Andrew Tridgell, πνευματικού πατέρα της SAMBA και πρωτεργάτη του rsync, ν' αναπτύξει ανοικτό λογισμικό το οποίο θα συνεργαζόταν με τα αποθετήρια του BitKeeper μέσω τεχνικών reverse engineering.

Ως αποτέλεσμα, ο Linus Torvalds ηγήθηκε της φιλόδοξης προσπάθειας για την ανάπτυξη μιας νέας πλατφόρμας διαχείρισης εκδόσεων. Επρόκειτο για το Git, ένα κατανεμημένο σύστημα που γράφτηκε μέσα σε δύο μόλις εβδομάδες. Πολύ γρήγορα, το Git εξελίχθηκε σε ένα αυτόνομο πρότζεκτ κι από νωρίς γνώρισε ευρεία αποδοχή μεταξύ των developers. Οι σχεδιαστικοί στόχοι του Git υπαγόρευαν την ανάπτυξη ενός απλού, γρήγορου και κατανεμημένου VCS, το οποίο θα ήταν ικανό να διαχειρίζεται χιλιάδες παράλληλα branches, να υποστηρίζει μεγάλα πρότζεκτ όπως εκείνο του Linux kernel, αλλά και να ενσωματώνει μηχανισμούς ασφαλείας από τις καταστροφές ή ακόμη κι από την εσκεμμένη δολιοφθορά. Από τη γέννησή του, το 2005, το Git κατάφερε να υλοποιήσει όλους αυτούς τους στόχους.

Χαρακτηριστικά που το κάνουν να ξεχωρίζει

Παραδοσιακά συστήματα ελέγχου εκδόσεων, όπως το CVS και το Subversion, βλέπουν τις πληροφορίες που διαχειρίζονται ως ακολουθίες αλλαγών σε επίπεδο αρχείου. Μ' άλλα λόγια, επιτηρούν ένα σύνολο αρχείων και για κάθε μέλος του συνόλου τηρούν μια ακολουθία αλλαγών, η οποία εξελίσσεται συν τω χρόνω. Προφανώς, οι ακολουθίες δεν εξελίσσονται με τον ίδιο ρυθμό, αφού δεν αλλάζουν όλα τα αρχεία με την ίδια συχνότητα. Κανένα VCS πάντως δεν έχει πρόβλημα μ' αυτό το φαινόμενο.

Το Git ακολουθεί μια εντελώς διαφορετική λογική: Θεωρεί τα δεδομένα που διαχειρίζεται ως μια ροή (stream) διακριτών στιγμιότυπων (snapshots) ενός μικρού συστήματος αρχείων (filesystem). Με το που ξεκινάμε ένα νέο πρότζεκτ στο Git, δημιουργούμε ουσιαστικά ένα νέο σύστημα αρχείων. Στο εξής, κάθε φορά που κάνουμε ένα commit ή αποθηκεύουμε την κατάσταση του πρότζεκτ, το Git είναι σαν να παίρνει μια φωτογραφία του όλου τοπίου και να την αποθηκεύει για μελλοντική αναφορά. Στην πραγματικότητα, αυτό που κάνει είναι περισσότερο αποδοτικό σε σύγκριση με ό,τι υπονοείται από την παρομοίωση με τη φωτογραφία. Πράγματι, αν τη στιγμή δημιουργίας στιγμιότυπου υπάρχουν αρχεία που δεν έχουν αλλάξει, τότε αντί να τα αποθηκεύσει στο νέο στιγμιότυπο δημιουργεί απλά συνδέσμους προς τις πιο πρόσφατες αποθηκευμένες εκδοχές των εν λόγω αρχείων.

Κατά την εργασία μας εξάλλου με το Git, όλες σχεδόν οι ενέργειες είναι τοπικές και σπανίως εμπλέκεται κάποιο απομακρυσμένο μηχάνημα. Αμέσως αμέσως αυτό σημαίνει ότι ξεχνάμε τον παράγοντα του network latency. Υποθέστε, για παράδειγμα, ότι θέλουμε να δούμε τις αλλαγές που έχουν συμβεί σε ένα αρχείο, συγκρίνοντας το σημερινό του περιεχόμενο μ' εκείνο που είχε τρεις εβδομάδες πριν. Χωρίς να επικοινωνήσει το Git με άλλον υπολογιστή, θα βρει το αρχείο όπως ήταν τρεις εβδομάδες πριν, αμέσως θα προχωρήσει σ' έναν υπολογισμό διαφορών τοπικά και, τέλος, θα μας τις εμφανίσει. Ακόμη λοιπόν κι αν βρεθούμε offline θα μπορούμε να δουλεύουμε στο πρότζεκτ και να κάνουμε τα commits μας και, αργότερα, όταν θα είμαστε και πάλι online, το πολύ πολύ να στείλουμε τις αλλαγές σε κάποιον απομακρυσμένο server.

Ό,τι πρέπει για να ξεκινήσετε

Αν θέλετε να μάθετε ένα VCS και να το κάνετε αναπόσπαστο μέρος της εργασίας σας, καλό είναι να γνωρίζετε ότι εκτός από την ταχύτητα και την ευελιξία το Git σας επιτρέπει να πειραματίζεστε με ασφάλεια. Οτιδήποτε κι αν κάνετε με το Git, ουσιαστικά προσθέτετε στοιχεία στη βάση δεδομένων του. Είναι εξαιρετικά δύσκολο να κάνετε κάτι μη αναστρέψιμο ή να χάσετε δεδομένα. Σίγουρα υπάρχει το ενδεχόμενο για μπερδέματα ή ακόμη και για απώλεια αρχείων *πριν* από το commit, μετά από κάθε commit όμως είναι δύσκολο να χάσετε κάτι – ειδικά όταν ανεβάζετε το πρότζεκτ σας σε απομακρυσμένο repository.

Σ' αυτό το πρώτο εισαγωγικό άρθρο συζητήσαμε πολλά και θεωρητικά. Πιστεύουμε ότι έχει έρθει η ώρα να ασχοληθούμε στην πράξη με το Git, ώστε να πάρουμε μια καλή ιδέα για τις δυνατότητές του. Προτείνουμε να συνεχίσετε μαζί μας, με το άρθρο που αρχίζει από τη σελίδα 54.

Skill: Intermediate
Tags: SQLite, SQL, databases, phpLiteAdmin

SQLite: Μια Βάση και στα Γρήγορα! [Μέρος 1/2]

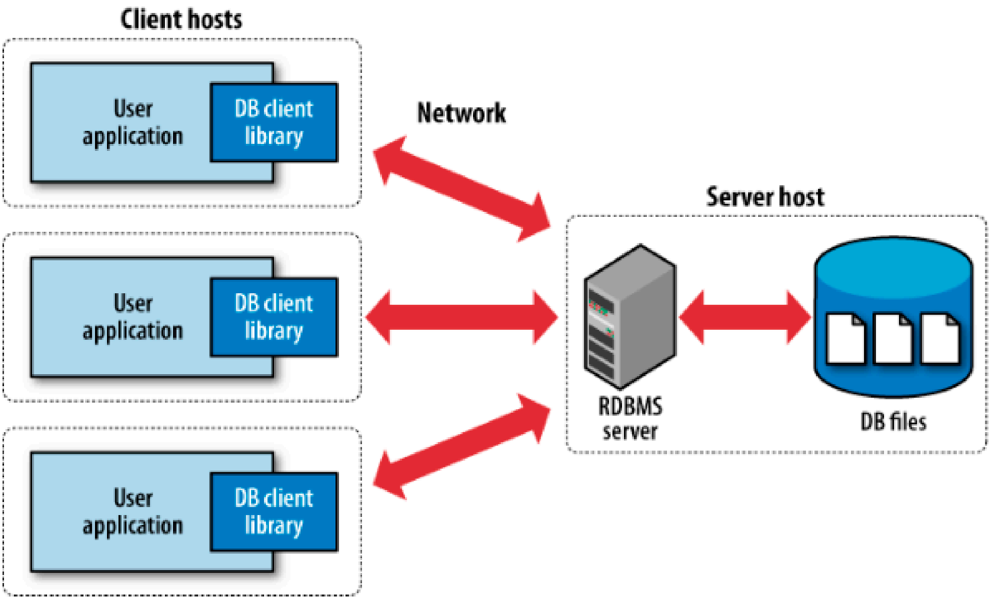
Πιστεύετε ότι η χρήση μιας βάσης δεδομένων για τις ανάγκες μικρών εφαρμογών αποτελεί περιττή πολυτέλεια; Σκεφτόσαστε μήπως ότι η εγκατάσταση ενός RDBMS σ' ένα σύστημα με περιορισμένους πόρους, θα του κάτσει βαρύ; Σωστά σκέπτεστε. Ευτυχώς που η SQLite καλύπτει άνετα την απόσταση μεταξύ της ισχύος ενός ολοκληρωμένου RDBMS, και της "ελαφρότητας" που απαιτούν οι σύγχρονες εφαρμογές.

του Πέτρου Κυλαδίτη

Σχεδόν κάθε φορά που αναπτύσσουμε μια εφαρμογή, ερχόμαστε αντιμέτωποι με το ζήτημα της αποθήκευσης και της ανάκτησης δεδομένων. Μπροστά σ' αυτό το πρόβλημα ενδέχεται να ακολουθήσουμε τη (φαινομενικά) απλή λύση της χρήσης μεμονωμένων αρχείων. Κάνοντας κάτι τέτοιο, όμως, αναλαμβάνουμε την ευθύνη να στήσουμε το μηχανισμό που θα διαχειρίζεται τα αρχεία και τα δεδομένα. Αυτό μπορεί να ακούγεται απλό, αλλά δεν είναι πάντα. Αν σκοπεύουμε να αποθηκεύουμε πολλά "κομματάκια" δεδομένων, αγνώστου "μήκους" και πλήθους, ο σωστός χειρισμός ενδέχεται ν' αποδειχθεί μεγάλος μπελάς. Αρκεί να σκεφτεί κανείς τα διάφορα bugs που μπορεί να ξεφυτρώσουν στον κώδικα, προκαλώντας μικρές και τυχαίες αλλοιώσεις στα δεδομένα ή μια συνολική επιβάρυνση στη λειτουργία του προγράμματος (π.χ., εξαιτίας κάποιων memory leaks). Τα παραπάνω προβλήματα πολλαπλασιάζονται όταν στον άγνωστο όγκο δεδομένων προσθέσουμε και τη διασύνδεσή τους με πολύπλοκες σχέσεις. Ειδικά σ' αυτές τις περιπτώσεις, η χρήση μιας βάσης δεδομένων αποτελεί την ιδανική επιλογή. Λύνει όλα μας τα προβλήματα εύκολα, εξασφαλίζει την ακεραιότητα των δεδομένων, ενώ κατά πάσα πιθανότητα θα βελτιώσει και τις επιδόσεις. Ωραία όλα αυτά, θα πείτε, αλλά πού είναι η παγίδα; Η χρήση μιας βάσης δεδομένων ή, πιο σωστά, ενός RDBMS (Relational Database Management System), προϋποθέτει την εγκατάσταση και τη ρύθμιση του αντίστοιχου server, ο οποίος θα εξυπηρετεί τα αιτήματα προς τη βάση. Μπορεί το (σωστό) στήσιμο τέτοιων υπηρεσιών να μην αποτελεί δύσκολη υπόθεση, σίγουρα όμως δεν ταιριάζει σε μια μικρή εφαρμογή που τρέχει σ' ένα σύστημα περιορισμένων πόρων και δυνατοτήτων.

Ο Δαβίδ των RDBMS

Όσα είπαμε προηγουμένως για τις βάσεις ισχύουν στις περισσότερες των περιπτώσεων, αλλά όχι πάντα. Η SQLite αποτελεί την εξαίρεση, καθώς πρόκειται για μια βάση δεδομένων που καλύπτει άνετα τις ανάγκες μιας μικρής εφαρμογής, ενώ προσφέρει τα προσόντα και τις ευκολίες που παρέχουν συνήθως τα μεγάλα συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων. Το δεύτερο συνθετικό του ονόματός της, το lite, δεν αναφέρεται στο πλήθος των δυνατοτήτων, αλλά στους πόρους που απαιτεί για τη λειτουργία της. Η SQLite δεν βασίζεται στη γνωστή αρχιτεκτονική client-server, που συναντάμε στα περισσότερα συστήματα του είδους. Πρόκειται για μια βιβλιοθήκη που διαχειρίζεται άμεσα ένα αρχείο δεδομένων, για κάθε βάση που δημιουργούμε. Δεν κάναμε λάθος: Η SQLite διατηρεί ένα μόλις αρχείο, την ίδια στιγμή που οι κλασικές βάσεις δημιουργούν και διαχειρίζονται ένα ολόκληρο *σύστημα* αρχείων. Αυτή η προσέγγιση προσφέρει αρκετά πλεονεκτήματα σε μια μικρή εφαρμογή, τα οποία σχετίζονται με τις απαιτήσεις σε πόρους και φυσικά με τις επιδόσεις. Επιπρόσθετα, οι διάφορες βάσεις της SQLite μεταφέρονται πανεύκολα, αφού αρκεί η αντιγραφή ενός μόνο αρχείου. Η βιβλιοθήκη SQLite μπορεί να ενσωματωθεί σε οποιαδήποτε εφαρμογή και σε κάθε λειτουργικό σύστημα, ενώ υποστηρίζει και το πρότυπο SQL (για την ακρίβεια το SQL92). Τέλος, αξίζει ν' αναφέρουμε ότι η εν λόγω βιβλιοθήκη παρέχεται εντελώς ελεύθερα, χωρίς ύπουλους περιορισμούς κι απαλλαγμένη από πατέντες.



Η πλειονότητα των καθιερωμένων RDBMS απαιτεί την ύπαρξη ενός server, σε ρόλο διαμεσολαβητή μεταξύ της εφαρμογής μας και της βάσης δεδομένων. Αυτό δεν ισχύει στην περίπτωση της SQLite!

Ανάλαφροι περιορισμοί

Θα έχετε μάλλον εκπλαγεί με το πόσα πολλά μπορεί να κάνει η SQLite και ίσως να αναρωτιέστε γιατί δεν είναι περισσότερο διαδομένη. Για την επίτευξη όσων αναφέραμε, η SQLite κάνει ορισμένες εκπτώσεις σε ζητήματα λειτουργικότητας, οι οποίες την καθιστούν ακατάλληλη για εφαρμογές μεγάλης κλίμακας. Κατ' αρχάς, το μέγεθος μιας βάσης στην SQLite υπόκειται στους περιορισμούς που θέτει για το μέγεθος των αρχείων το εκάστοτε file system. Ένα άλλο μειονέκτημα σχετίζεται με τα σενάρια χρήσης στα οποία απαιτείται διαβαθμισμένη πρόσβαση στη βάση. Η SQLite δεν επιτρέπει σε ξεχωριστά προγράμματα –ούτε σε επιμέρους υποσυστήματα της ίδιας εφαρμογής– να διαβάζουν τη βάση με διαφορετικά δικαιώματα. Γενικότερα, υποστηρίζει μόνο ένα "κλείδωμα" για κάθε βάση, το οποίο αφορά σε όλους τους πίνακες. Στο συγκεκριμένο ζήτημα, η SQLite ακολουθεί τη λογική "ή όλα ή τίποτα". Επιπρόσθετα, η SQLite παρουσιάζει μια σχετική ακαμψία σε ό,τι αφορά στη δομή των πινάκων. Έτσι, ενώ είναι πολύ εύκολο να προσθέσουμε νέα πεδία (νέες στήλες) στο τέλος ενός πίνακα, είναι αδύνατο να διαγράψουμε κάποιο από τα προϋπάρχοντα πεδία. Αυτό σημαίνει ότι όταν σχεδιάζουμε βάσεις δεδομένων για την SQLite πρέπει να επιδεικνύουμε μεγάλη προσοχή και προνοητικότητα, ώστε αργότερα να μη χρειάζονται αλλαγές. Πάντως, ακόμα κι αν προκύψει η ανάγκη για μεγάλες αλλαγές σε κάποιον πίνακα, δεν θα έρθει το τέλος του κόσμου. Μπορούμε να φτιάξουμε έναν νέο πίνακα με τη σωστή δομή και να αντιγράψουμε από τον παλιό μόνο τα πεδία που χρειαζόμαστε. Τέλος, ένα ακόμα χαρακτηριστικό που απουσιάζει από το οπλοστάσιο της SQLite είναι η δυνατότητα δημιουργίας ρεπλίκας της βάσης, σε πραγματικό χρόνο.

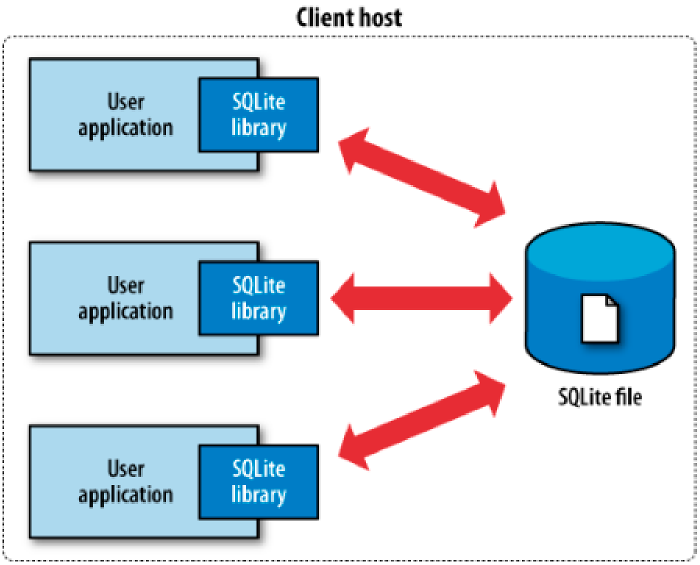
Θέλουμε να πιστεύουμε ότι οι περιορισμοί που αναφέραμε δεν σας απογοήτευσαν. Η SQLite είναι μικρή, ελαφριά και παρ' όλο που θα μπορούσε να υποκαταστήσει κάποια από τις "μεγάλες" βάσεις, μπορεί να αξιοποιηθεί σε *πάρα πολλές* μικρές εφαρμογές.

Στα ενδότερα της SQLite

Οι διαφορές που παρουσιάζει η SQLite έναντι των υπολοίπων RDBMS δεν εξαντλούνται σε όσα είπαμε ήδη. Στα άλλα συστήματα, κάθε στήλη (πεδίο) ενός πίνακα μπορεί να φιλοξενεί δεδομένα συγκεκριμένου τύπου. Για παράδειγμα, μόνο κείμενο (text), μόνο αριθμούς κινητής υποδιαστολής (float), μόνο ακεραίους (integer) και πάει λέγοντας. Οι πίνακες της SQLite παρουσιάζουν μεγαλύτερη ευελιξία σ' αυτόν τον τομέα. Μια στήλη μπορεί να φιλοξενεί δεδομένα διαφορετικών τύπων, αρκεί αυτοί οι τύποι να ανήκουν στην ίδια οικογένεια. Έτσι, κατά την προσθήκη ενός πεδίου σε έναν πίνακα δεν ορίζουμε κάποιο συγκεκριμένο τύπο αλλά το "βαθμό συγγένειας" (affinity) των δεδομένων που θα φιλοξενούνται. Αυτή η δήλωση δεν είναι δεσμευτική για το μέλλον, αλλά βοηθάει την SQLite να κάνει τις κατάλληλες μετατροπές καθώς εισάγουμε δεδομένα στον πίνακα. Οι επιλογές που προσφέρονται εδώ είναι οι εξής: NULL, INTEGER, REAL, TEXT και BLOB. Νομίζουμε ότι δεν χρειάζεται να δώσουμε ιδιαίτερες εξηγήσεις για τον καθένα. Μόνο ο τύπος BLOB (Binary Large Object) αξίζει ιδιαίτερης αναφοράς, καθώς είναι ιδανικός για την αποθήκευση ολό-

κληρων αρχείων εικόνας, ήχου ή και προγραμμάτων. Σημειώστε ότι σε κάθε πίνακα υπάρχει ένα πεδίο του οποίου ο τύπος δεν μπορεί να αλλάξει. Πρόκειται για το πεδίο INTEGER PRIMARY KEY, που φιλοξενεί πάντα μοναδικούς ακεραίους αριθμούς.

Δεν ξέρουμε αν το προσέξατε, αλλά η SQLite δεν διαθέτει κάποιον τύπο για την αποθήκευση ημερομηνίας και ώρας. Αυτό σημαίνει ότι μπορούμε να αξιοποιήσουμε τον τύπο TEXT και να προσδιορίσουμε την εκάστοτε χρονική στιγμή με οποιονδήποτε τρόπο μας εξυπηρετεί. Εναλλακτικά, μπορούμε να χρησιμοποιήσουμε τον τύπο INTEGER και να εκφράσουμε την ημερομηνία με τη μορφή ενός timestamp, όπως γίνεται στα συστήματα Unix. Ένας άλλος διαδεδομένος τύπος που απουσιάζει από την SQLite είναι ο Boolean. Η λύση είναι και πάλι απλούστατη, αφού αρκεί να ορίσουμε μια σύμβαση του τύπου "το 0 αντιστοιχεί στην κατάσταση FALSE και το 1 στην κατάσταση TRUE".



Με τη βοήθεια της βιβλιοθήκης SQLite οι εφαρμογές επικοινωνούν με τις βάσεις δεδομένων σε τοπικό επίπεδο, χωρίς τη μεσολάβηση κάποιου άλλου προγράμματος ή κάποιας δικτυακής σύνδεσης.

Εργασία με την SQLite

Για τη χρήση της SQLite μπορούμε να χρησιμοποιήσουμε τον package manager της διανομής μας (αν χρησιμοποιούμε Linux) ή να μεταβούμε στην περιοχή Downloads του επίσημου site (www.sqlite.org) και να κατεβάσουμε το επιθυμητό πακέτο για το λειτουργικό μας. Η SQLite διατίθεται ως βιβλιοθήκη για όλα τα δημοφιλή συστήματα (desktop και mobile), ενώ προσφέρεται και στη μορφή του πηγαίου κώδικα (είναι γραμμένη σε C), για την εύκολη ενσωμάτωση σε δικά μας έργα. Επιπρόσθετα προσφέρεται κι ένα εργαλείο για τη γραμμή εντολών, το οποίο μοιάζει με το "τερματικό" της MySQL. Εμείς κατεβάσαμε το σχετικό ZIP, το αποσυμπιέσαμε κι αφού ανοίξαμε μια κονσόλα εκτελέσαμε το sqlite3 ως εξής:

```
sqlite3 mybase.db
SQLite version 3.13.0 2016-05-18 10:57:30
Enter ".help" for usage hints.
sqlite>_
```

Όπως καταλαβαίνετε, το mybase.db αποτελεί το αρχείο με το οποίο θέλουμε να εργαστούμε. Εάν το αρχείο δεν υπάρχει, το πρόγραμμα της sqlite θα το δημιουργήσει αυτόματα. Σημειώστε επίσης ότι η επέκταση ".db" είναι προαιρετική και τη χρησιμοποιήσαμε αποκλειστικά για τη δική μας ευκολία. Θα μπορούσαμε να παραλείψουμε το όνομα του αρχείου, οπότε κάθε εργασία που θα εκτελούσαμε στη συνέχεια θα αποθηκευόταν σε μια προσωρινή βάση δεδομένων, στη μνήμη. Μια ειδοποιός διαφορά σε σχέση με το αντίστοιχο περιβάλλον της MySQL αφορά στη μορφή των εντολών. Στην περίπτωση της SQLite ξεκινούν όλες με μία τελεία. Έτσι, για να εμφανιστεί μια λίστα με τις διαθέσιμες εντολές αρκεί να δώσουμε

```
.help
```

Όπως θα διαπιστώσετε, η λίστα που τυπώνεται είναι μακροσκελής και η παράθεσή της εδώ δεν θα προσέφερε απολύτως τίποτα. Στη συνέχεια θα δούμε μερικές μόνο από τις βασικές εντολές, ενώ αργότερα, εξετάζοντας τη δημιουργία μιας μικρής εφαρμογής, θα δούμε περισσότερες.

```
Διαχειριστής: Γραμμή εντολών - sqlite3.exe mybase.db
sqlite> .tables
sqlite> CREATE TABLE telcat(id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,onoma
TEXT,tel TEXT,city TEXT);
sqlite> .tables
telcat
sqlite> INSERT INTO telcat (onoma, tel, city) VALUES
   ...>    ('PROKOMENOS', '03123456','SALONIKA'),
   ...>    ('TAKIS', '69345678', 'AVOROS'),
   ...>    ('KOSTAKIS', '21567893', 'PERA RAHOULA')
   ...> ;
sqlite> SELECT * FROM telcat ;
1|PROKOMENOS|03123456|SALONIKA
2|TAKIS|69345678|AVOROS
3|KOSTAKIS|21567893|PERA RAHOULA
sqlite> .header on
sqlite> .mode column
sqlite> .timer on
sqlite> SELECT * FROM telcat ;
id          onoma       tel         city
----------  ----------  ----------  ----------
1           PROKOMENOS  03123456    SALONIKA
2           TAKIS       69345678    AVOROS
3           KOSTAKIS    21567893    PERA RAHOU
Run Time: real 0.016 user 0.000000 sys 0.000000
sqlite>
```

Το λιτό περιβάλλον του εργαλείου sqlite3. Η εντολή .mode μπορεί να τροποποιήσει δραματικά την έξοδο από την εκτέλεση ερωτημάτων SQL.

Για να δούμε όλες τις βάσεις με τις οποίες εργαζόμαστε, αρκεί να εκτελέσουμε την εντολή .databases, ενώ για να δημιουργήσουμε άμεσα ένα αντίγραφο του ανοιγμένου αρχείου μπορούμε να χρησιμοποιήσουμε την εντολή .clone. Για να εκτελέσουμε ένα ερώτημα SQL, το οποίο βρίσκεται αποθηκευμένο σε κάποιο αρχείο κειμένου, προσφέρεται η εντολή .read. Αν πάλι εργαζόμαστε με μια βάση δεδομένων που βρίσκεται στη RAM, μπορούμε να τη μεταφέρουμε σε αρχείο με τη βοήθεια της εντολής .save. Ας εξετάσουμε τώρα και κάτι πιο χρήσιμο. Θα δημιουργήσουμε έναν νέο πίνακα! Ο πίνακας ονομάζεται telcat και, όπως υποψιάζεστε, θα χρησιμοποιηθεί σαν ένας απλούστατος τηλεφωνικός κατάλογος. Εκτός από το αναγνωριστικό πεδίο (πρωτεύον κλειδί) ο πίνακας θα περιλαμβάνει και τρία πεδία κειμένου, για το όνομα, το τηλέφωνο και την πόλη κατοικίας κάθε επαφής. Το ερώτημα SQL που δημιουργεί αυτόν τον πίνακα έχει την ακόλουθη μορφή:

```
sqlite> CREATE TABLE telcat(
   ...>   id INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
   ...>   onoma TEXT,
   ...>   tel TEXT,
   ...>   city TEXT
   ...> );
sqlite>
```

Σημειώστε ότι η εντολή create δεν έχει τελεία. Αυτό συμβαίνει διότι ο κανόνας που αναφέραμε με τις τελείες αφορά μόνο στις εντολές της SQLite και σε εκείνες της SQL. Ερωτήματα όπως το παραπάνω, που έχουν σχετικά μακροσκελή έκφραση και τα οποία θα θέλαμε να χρησιμοποιήσουμε πολλές φορές με ελάχιστες αλλαγές, μπορούμε να τα τοποθετούμε σε αρχεία κειμένου και να τα φορτώνουμε με την εντολή .read. Αν είχαμε δημιουργήσει ένα τέτοιο αρχείο με το όνομα create_table.txt, θα μπορούσαμε να εκτελέσουμε το ερώτημα που περιλαμβάνει ως εξής:

```
sqlite> .read create_table.txt
```

Για να βεβαιωθούμε τώρα ότι ο πίνακας δημιουργήθηκε μπορούμε να χρησιμοποιήσουμε την εντολή .tables, η οποία εμφανίζει όλους τους πίνακες της τρέχουσας βάσης δεδομένων. Στην παρούσα φάση θα πρέπει να δούμε μόνο έναν και με το όνομα telcat.

Ας εισάγουμε τώρα και μερικά δεδομένα, αξιοποιώντας πάλι την εντολή .read κι ένα αρχείο κειμένου. Αυτή τη φορά δημιουργούμε το αρχείο data.txt και εισάγουμε τα ακόλουθα:

```
INSERT INTO telcat (onoma, tel, city) VALUES
  ('PROKOMMENOS', '0933123456','SALONIKA'),
  ('TAKIS', '0963678848', 'AVOROS'),
  ('KOSTAKIS', '231956783', 'PERA RAHOULA')
;
```

Στη συνέχεια, από τη γραμμή εντολών της SQLite δίνουμε το εξής:

```
sqlite> .read data.txt
```

Αν πάνε όλα καλά, το πρόγραμμα δεν θα εμφανίσει κανένα μήνυμα λάθους ούτε και κάτι άλλο. Η γραμμή εντολών της SQLite είναι αρκετά σιωπηρή και προτιμά να μη μας κουράζει με περιττά μηνύματα, εκτός κι αν συμβαίνει κάτι πολύ σημαντικό! Για να τσεκάρουμε τα δεδομένα που έχουμε εισαγάγει προηγουμένως στον πίνακα, αρκεί να εκτελέσουμε ένα νέο ερώτημα SQL:

```
sqlite> SELECT * FROM telcat ;
1|PROKOMMENOS |03123456|SALONIKA
2|TAKIS|69345678|AVOROS
3|KOSTAKIS|21567893|PERA RAHOULA
```

Όπως βλέπετε, κάθε εγγραφή εμφανίζεται σε ξεχωριστή γραμμή, αλλά οι στήλες δεν είναι στοιχημένες. Το αποτέλεσμα που προκύπτει από την εκτέλεση του ερωτήματος μοιάζει χαοτικό, σε αντίθεση με το τακτοποιημένο output που μας έχει συνηθίσει η γραμμή εντολών της MySQL. Φυσικά αυτό δεν αποτελεί ουσιαστικό μειονέκτημα για μια εφαρμογή που χρησιμοποιεί τη βάση, αλλά ανήκει σ' εκείνες τις ελλείψεις που καθιστούν την SQLite τόσο απλή και μικρή. Πάντως, για όσους συνηθίζουν να εργάζονται από τη γραμμή εντολών, ο τρόπος παρουσίασης των πινάκων μπορεί να βελτιωθεί αρκετά. Αρκεί να τροποποιήσουμε ορισμένες παραμέτρους. Δείτε τις ακόλουθες γραμμές:

```
sqlite> .header on
sqlite> .mode column
sqlite> .timer on
sqlite>
sqlite> SELECT * FROM telcat ;
```

```
id          onoma       tel         city
----------  ----------  ----------  ----------
1           PROKOMMENOS 03123456    SALONIKA
2           TAKIS       69345678    AVOROS
3           KOSTAKIS    21567893    PERA RAHOULA
Run Time: real 0.016 user 0.000000 sys 0.000000
```

Η εντολή .mode καθορίζει τον τρόπο προβολής κι εκτός από την παράμετρο column, μπορεί να δεχτεί τις εξής: ascii, csv, html, insert, line, list, tabs και tcl. Ιδιαίτερα ενδιαφέρουσες είναι οι csv, html και insert. Η πρώτη εμφανίζει τα δεδομένα σε μορφή κειμένου και κάθε πεδίο διακρίνεται από τα άλλα με ένα κόμμα. Η δεύτερη τοποθετεί τα δεδομένα σε πίνακες HTML, ενώ η τρίτη δημιουργεί ερωτήματα SQL που αν τα εκτελούσαμε θα είχαν σαν αποτέλεσμα την εισαγωγή των αντίστοιχων δεδομένων.

Εργαλεία διαχείρισης

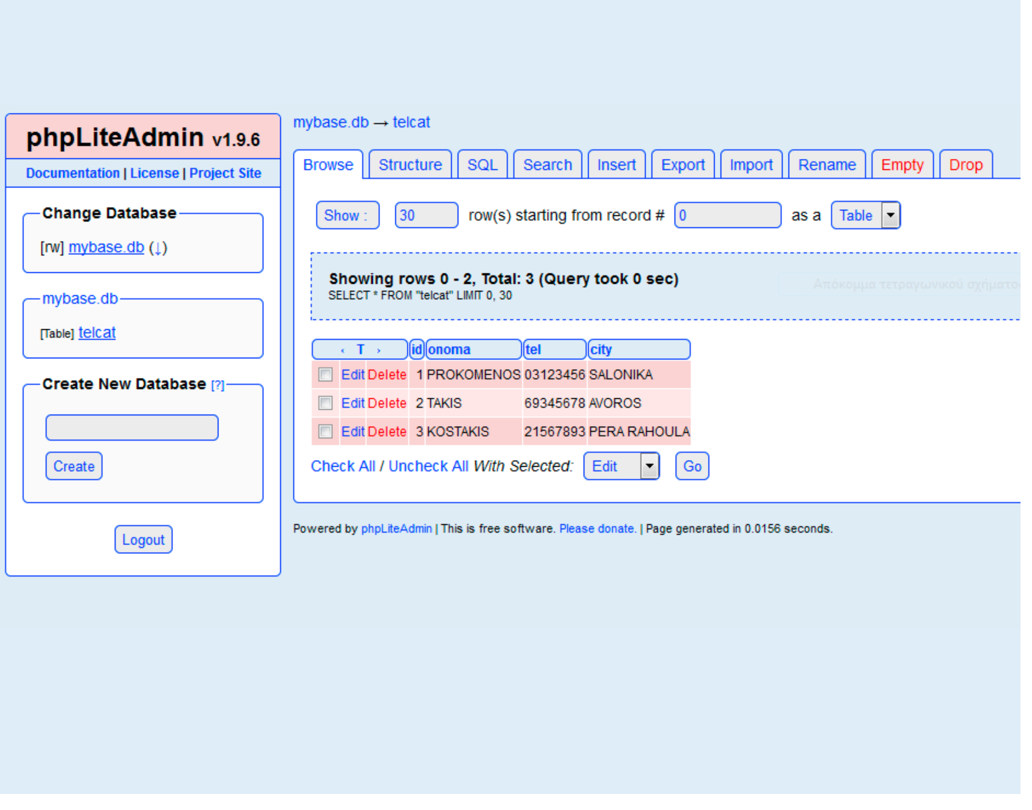
Από τη γραμμή εντολών της SQLite μπορούμε να κάνουμε τα πάντα. Για πολλούς χρήστες ωστόσο αυτός ο τρόπος εργασίας είναι κατάλληλος μόνο για μερικούς γρήγορους ελέγχους και για άλλες απλές δουλειές. Αυτό δεν αποτελεί πρόβλημα. Υπάρχουν πάμπολλα εργαλεία για τη διαχείριση βάσεων της SQLite, που διαθέτουν είτε παραθυρικό περιβάλλον είτε web interface. Εμείς, επειδή το phpMyAdmin μάς έχει καλομάθει, αποφασίσαμε να δούμε το αντίστοιχο εργαλείο για την SQLite, το οποίο ονομάζεται phpLiteAdmin (www.phpliteadmin.org).

Αν κατεβάσετε το πακέτο με την πιο πρόσφατη έκδοση του προγράμματος, θα βρείτε εντός του τρία αρχεία. Το ένα περιέχει μερικές γενικές πληροφορίες (readme.md), ενώ το phpliteadmin.php αποτελεί την ίδια την εφαρμογή. Το αρχείο phpliteadmin.config.sample.php αποτελεί ένα προαιρετικό αρχείο ρυθμίσεων. Λέμε ότι είναι προαιρετικό διότι οι διαθέσιμες ρυθμίσεις μπορούν να γίνουν και απευθείας, μέσα στο phpliteadmin.php. Επιλέξαμε να χρησιμοποιήσουμε το

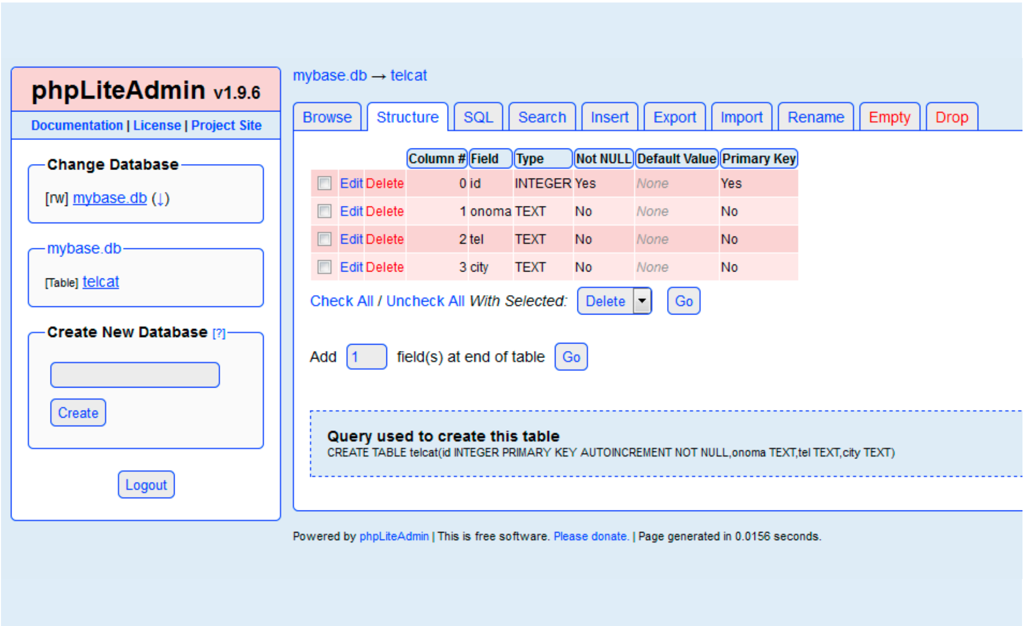
αρχείο ρυθμίσεων, όπως θα έκανε και κάθε σώφρων άνθρωπος κατά την ταπεινή μας γνώμη :D Μέσα στο αρχείο βρήκαμε τη μεταβλητή $password, στην οποία δώσαμε τον επιθυμητό κωδικό για την πρόσβαση στην εφαρμογή. Ο συγκεκριμένος κωδικός αφορά στο phpLiteAdmin και μόνο, αφού ούτως ή άλλως το σύστημα της SQLite δεν υποστηρίζει προστασία με κωδικό. Η μεταβλητή $directory δηλώνει τον κατάλογο εργασίας και είναι ορισμένη με τέτοιον τρόπο, ώστε να παραπέμπει στη θέση του ίδιου του phpLiteAdmin. Περιττό να πούμε ότι αυτό δεν μας άρεσε καθόλου. Για να μπορούμε να χρησιμοποιήσουμε το phpLiteAdmin, το αρχείο phpliteadmin.php πρέπει να βρίσκεται μέσα στο Document Root του web server που χρησιμοποιούμε. Αντιθέτως, τα αρχεία με τις βάσεις δεδομένων είναι φρόνιμο να βρίσκονται

σε κάποια θέση *έξω* από το Document Root. Έτσι, τροποποιήσαμε κατάλληλα την τιμή της μεταβλητής $directory, ώστε ο κατάλογος εργασίας του προγράμματος να βρίσκεται σε κάποια ασφαλή τοποθεσία.

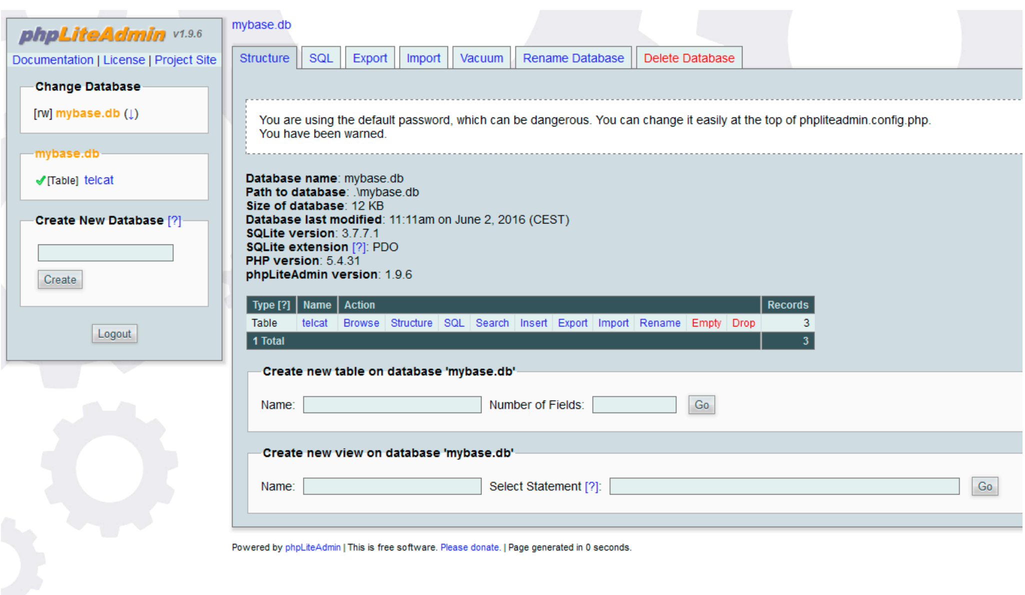
Η επόμενη μεταβλητή που συναντάμε στο αρχείο ρυθμίσεων είναι η $subdirectories, που αν οριστεί σε true επιβάλει την αυτόματη σάρωση των υποκαταλόγων για τον εντοπισμό κι άλλων βάσεων. Οι μεταβλητές που ακολουθούν αφορούν κατά κύριο λόγο στην όψη του προγράμματος και δεν χρειάζεται να πούμε περισσότερα. Τέλος, να σημειώσουμε ότι για να ληφθούν υπόψη οι ρυθμίσεις μας πρέπει να μετονομάσουμε το αρχείο ρυθμίσεων αφαιρώντας από το όνομά του το ".sample" (phpliteadmin.config.php). Για τον τρόπο χρήσης του phpLiteAdmin δεν χρειάζεται να πούμε τίποτα, αφού είναι παρόμοιος με εκείνον του δημοφιλούς phpMyAdmin. Στο επόμενο μέρος του μίνι αφιερώματός μας θα αξιοποιήσουμε την SQLite προγραμματιστικά, μέσα σε μια δική μας εφαρμογή. Έτσι θα αποκτήσουμε μια σφαιρική εικόνα για όσα προσφέρει το εν λόγω RDBMS και θα εμπλουτίσουμε το προγραμματιστικό μας οπλοστάσιο με ένα ακόμα ισχυρό εργαλείο.



Το phpLiteAdmin απλοποιεί σε μεγάλο βαθμό τη διαχείριση μιας βάσης δεδομένων SQLite. Ειδικά αν έχετε συνηθίσει να εργάζεστε με το phpMyAdmin, το περιβάλλον του phpLiteAdmin θα σας φανεί εξαιρετικά απλό και οικείο.



Οι αλλαγές που μπορούμε να κάνουμε σ' έναν ήδη σχηματισμένο πίνακα, σε μια βάση της SQLite, είναι περιορισμένες. Μέσα από το περιβάλλον του phpLiteAdmin, αυτές οι εργασίες απλοποιούνται ακόμα περισσότερο.



Το phpLiteAdmin συνοδεύεται από διάφορα θέματα που τροποποιούν την εμφάνισή του. Ένα από αυτά είναι φανερά εμπνευσμένο από την όψη του phpMyAdmin.

Skill: Intermediate
Tags: game development, Unity, textures, z-buffer, z-fighting

Όταν Μαλώνουν τα Textures

Το παιχνίδι μας έχει αρχίσει να παίρνει μορφή. Σε αυτό το άρθρο θα ολοκληρώσουμε την τοποθεσία των γεννητριών εμποδίων και θα προσθέσουμε διακόσμηση στην πίστα, έτσι, για να σπάει η μονοτονία.

του Γιάννη Κρομμύδα

Αρχικά θέλουμε να σας ζητήσουμε μια συγγνώμη. Σας υποσχεθήκαμε να ασχοληθούμε με physics σ' αυτό το τεύχος, αλλά τελικά θα χρειαστεί να το αναβάλουμε για κάποιο επόμενο (ίσως και για *το* επόμενο). Για το λόγο αυτό, η θεωρία του παρόντος άρθρου θα είναι ελάχιστη.

Δεν ξέρουμε αν έχετε ήδη τοποθετήσει επιπλέον spawners στη δική σας εκδοχή του παιχνιδιού. Όπως και να ΄χει, θα σας πούμε πώς γίνεται και ποιες πρέπει να είναι οι θέσεις τους, ώστε να καλύψουμε όλη την επιφάνεια που θα δραστηριοιείται ο παίχτης.

Στο παράθυρο Hierarchy του Unity επιλέγουμε το αντικείμενο Spawner και πατάμε [Ctrl+D], για να δημιουργήσουμε ένα αντίγραφό του. Το τοποθετούμε στο σημείο (-5, 0.5, -5.5), θέτοντας τις αντίστοιχες τιμές στα πεδία Position του Transform του. Επαναλαμβάνουμε τη διαδικασία δημιουργώντας άλλα επτά αντίγραφα. Οι θέσεις των spawners στην πίστα, με πρώτον αυτόν που βρίσκεται πιο κοντά στη σφαίρα του παίχτη (και προς τα πάνω, όπως βλέπουμε την πίστα) είναι οι ακόλουθες:

1. (5, 0.5, -7.5)
2. (-5, 0.5, -5.5)
3. (5, 0.5, -1,5)
4. (-5, 0.5, 0.5)
5. (5, 0.5, 2.5)
6. (-5, 0.5, 6.5)
7. (5, 0.5, 8.5)
8. (-5, 0.5, 10.5)
9. (5, 0.5, 12.5)

Λογικά θα παρατηρήσατε την αλληλουχία των αριθμών. Θα ήταν πολύ εύκολο να κάνουμε ένα script που θα τοποθετεί τους spawners αυτόματα στις θέσεις τους και τυχαία αριστερά ή δεξιά της πίστας, ώστε να δώσουμε ακόμη μια νότα τυχαιότητας στη δημιουργία της πίστας. Θ' αφήσουμε όμως τέτοιες αλλαγές για το polishing του παιχνιδιού, στο τέλος.

Παρατηρώντας το Hierarchy, πιστεύουμε ότι οι spawners θα ήταν καλύτερα αν βρίσκονταν συμμαζεμένοι σε ένα GameObject. Θα αποφύγουμε επιπλέον το clutter του Hierarchy σε περίπτωση που ψάχνουμε να βρούμε κάτι (και δεν χρησιμοποιούμε το search).

- Δημιουργούμε ένα νέο κενό GameObject στην πίστα, με [Ctrl+Shift+N]
- Το ονομάζουμε "Spawners"
- Το τοποθετούμε στο σημείο (0, 0, 0) και
- σέρνουμε όλα τα αντικείμενα Spawner της πίστας μας σε αυτό. Τα μαρκάρουμε όλα με το [Ctrl] και τα μεταφέρουμε στο άδειο αντικείμενο, ώστε να γίνουν παιδιά του.
- Έπειτα κάνουμε κλικ στο βελάκι αριστερά του αντικειμένου Spawners, ώστε να τα κρύψουμε. Αν ποτέ τα χρειαστούμε, από αυτό το βελάκι μπορούμε να τα εμφανίσουμε.
- Επιλέγοντας όλα τα αριστερά αντικείμενα Spawner από τη σκηνή (ή από το Hierarchy), στο component Spawner που εμφανίζεται θέτουμε το Speed=1 στον Inspector.
- Αντίστοιχα, σε όλα τα αντικείμενα στα δεξιά της πίστας, το Speed πρέπει να είναι ίσο με -1.

Αν τρέξουμε το παιχνίδι μας, θα δούμε ότι πλέον η πίστα γεμίζει με εμπόδια που κινούνται αριστερά και δεξιά.

Πίστα και Διακόσμηση

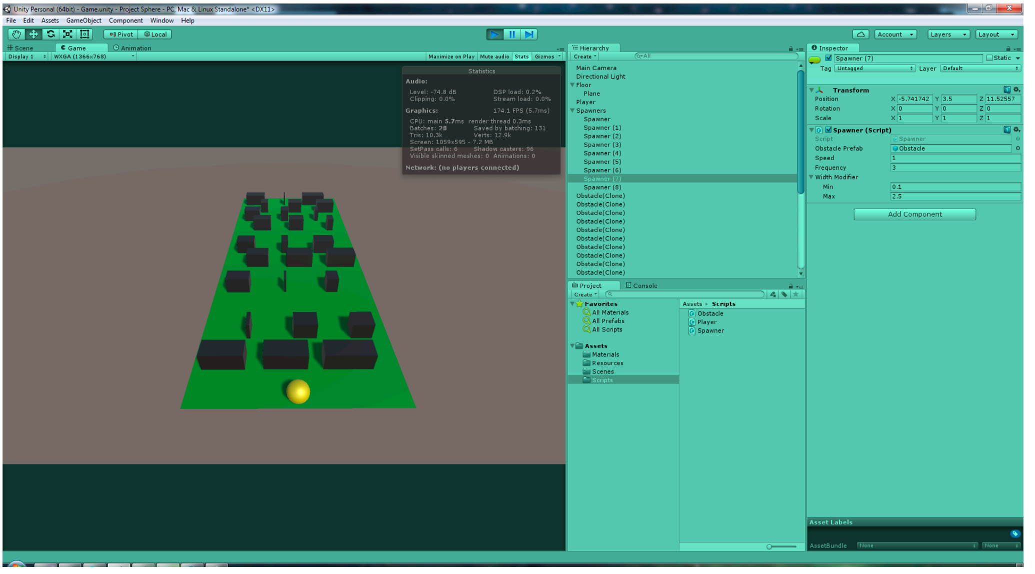
Βλέποντας την πίστα, έχουμε την εντύπωση ότι είναι λίγο μονότονη. Πάμε να σπάσουμε αυτή την πρασινίλα της, προσθέτοντας μερικά διακοσμητικά props. Αν θυμόσαστε από το game design document των τριών παραγράφων στο άρθρο που ξεκινήσαμε τη δημιουργία του παιχνιδιού, είχαμε πει ότι θα χωρίσουμε τα εμπόδια σε ομάδες. Θα έχουμε κάποιες νεκρές ζώνες, χρώματος λευκού, ώστε να διευκολύνουμε το χρήστη. Στο τέλος της πίστας, προκειμένου να σηματοδοτήσουμε τον τερματισμό θα χρησιμοποιήσουμε μια ασπρόμαυρη καρό γραμμή. Σαν τη σημαία που χρησιμοποιούν στην Formula 1.

Ας ξεκινήσουμε με τις λευκές γραμμές.

- Δημιουργούμε ένα νέο Plane με δεξί κλικ στο Hierarchy και 3D Object → Plane.
- Το ονομάζουμε RestZone και
- θέτουμε Scale = (1, 1, 0.1)
- Το τοποθετούμε στο σημείο (0, 0, -3.5)
- Με [Ctrl+D] κάνουμε ένα αντίγραφο και το τοποθετούμε στο σημείο (0, 0, 4.5)

Αυτές είναι οι δύο νεκρές ζώνες.

Για να δημιουργήσουμε τη γραμμή του τερματισμού, θα χρειαστεί να κάνουμε το texture της. Στο Paint (!), δημιουργούμε ένα αρχείο PNG μεγέθους 128x128 pixels. Ζωγραφίζουμε ένα μαύρο τετράγωνο 64x64 pixels πάνω αριστερά κι άλλο ένα κάτω



Μερικές φορές ο αλγόριθμος παραγωγής τυχαίων αριθμών δεν κάνει και πολύ καλά τη δουλειά του. Υπάρχουν και εναλλακτικές, όμως.

δεξιά του κέντρου της εικόνας. Το αποθηκεύουμε ως checkered.png στον φάκελο "Assets/Materials" του project μας. Με το που θα γυρίσουμε στον Editor, η Unity θα έχει ήδη διαβάσει το texture, θα το έχει εισαγάγει στο παιχνίδι μας και θα μπορούμε να το δούμε στο παράθυρο Project.

Η μηχανή βλέπει το αρχείο ως Texture, αλλά για να το χρησιμοποιήσουμε σε κάποιο μοντέλο πρέπει να δημιουργήσουμε ένα material μ' αυτό.

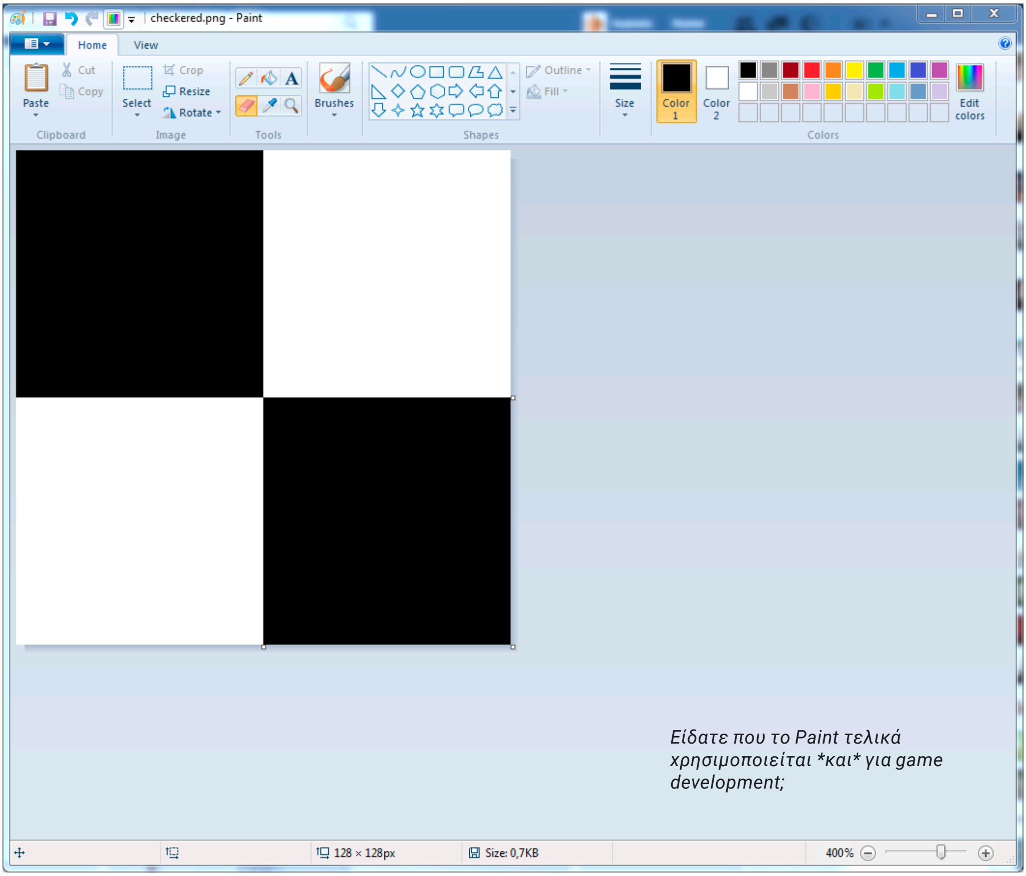
- Στο φάκελο Assets/Materials του project, κάνοντας δεξί κλικ κι επιλέγουμε Create → Material.
- Το νέο meterial τ' ονομάζουμε "FinishLine" και
- σέρνουμε το texture "checkered" από το Project στον Inspector, στο κουτί που βρίσκεται αριστερά της ιδιότητας Albedo του νέου material που έχουμε δημιουργήσει.

Το material είναι έτοιμο, οπότε πάμε να δημιουργήσουμε τη γραμμή τερματισμού.

- Στο Hierarchy κάνουμε ακόμα ένα αντίγραφο του RestZone (με [Ctrl+D]).
- Το ονομάζουμε FinishLine και
- το τοποθετούμε στο σημείο (0, 0, 14.5).
- Βρίσκουμε το component με όνομα Mesh Rendered, επεκτείνουμε τη λίστα "Materials" και θέτουμε σαν "Element 0" το material FinishLine από τα assets μας.

Η πίστα δεν χωράει σε μήκος όλα τα στοιχεία που της βάλαμε, οπότε πρέπει να κάνουμε κάποιες αλλαγές και σε αυτή. Με επιλεγμένο το αντικείμενο Floor στο Hierarchy, αλλάζουμε το Position του σε (0, 0, 2.5) και το Scale του σε (1, 1, 2.5).

Έτσι, η πίστα μας θα έχει αρκετό μήκος ώστε να καλύψει όλη την απόσταση που πρέπει να διανύσει ο χρήστης. Σε αυτό το σημείο η οθόνη σας πρέπει να φαίνεται όπως δείχνει η εικόνα παρακάτω.

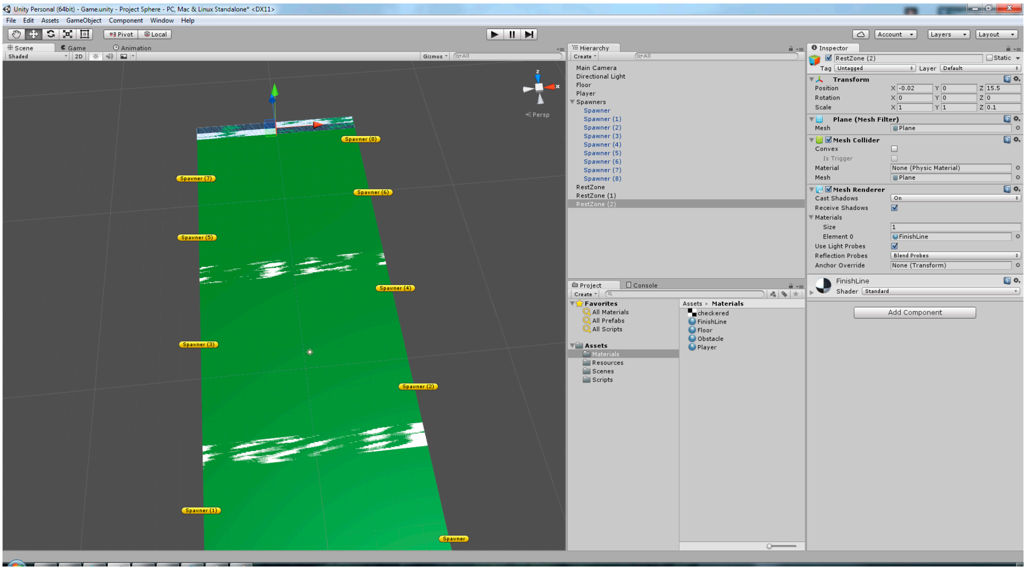


Τι θα γίνει render τελικά;

Είναι φανερό ότι κάτι δεν πάει καλά με τα textures των γραμμών που έχουμε βάλει στην πίστα. Το φαινόμενο αυτό ονομάζεται Z-Fighting και παρουσιάζεται όταν δύο επιφάνειες έχουν παρόμοιες τιμές στο z-buffer.

Σε μια σκηνή (κι όχι μόνο) είναι λογικό κάποια αντικείμενα να βρίσκονται μπροστά ή πίσω από κάποια άλλα. Η φυσιολογία του ματιού του επιτρέπει να διακρίνει ό,τι βρίσκεται πιο μπροστά (ξεχάστε τα αντικείμενα που είναι διάφανα ή ημιδιάφανα για λίγο). Ο αλγόριθμος του rendering της κάρτας γραφικών πάνω κάτω κάνει τη δουλειά του ματιού μας. Πρέπει να ζωγραφίσει μια ολοκληρωμένη εικόνα στην οθόνη μας, όπως θα τη βλέπαμε και με τα μάτια μας αν ήταν πραγματικότητα. Από τη φύση του, ο αλγόριθμος δεν κάνει διακρίσεις στο τι πρέπει να προβληθεί ή όχι. Όποιο αντικείμενο βρίσκεται μέσα στο viewport πρέπει να προβληθεί, αλλά αυτό δεν σημαίνει ότι πρέπει να είναι κι ορατό από το χρήστη.

Όταν ένα αντικείμενο πρέπει να προβληθεί στην οθόνη μας και μια και η εικόνα που βλέπουμε είναι δύο διαστάσεων, το βάθος κάθε pixel (δηλαδή η τρίτη διάσταση) που ανήκει στο αντικείμενο αυτό, αποθηκεύεται σε ένα προσωρινό πίνακα δύο



Z-Fighting, όταν τα textures προσπαθούν να αλληλεπικαλυφθούν.

διαστάσεων. Ο πίνακας αυτός ονομάζεται z-buffer ή depth buffer. Όταν λοιπόν περισσότερα από ένα αντικείμενα πρέπει να προβληθούν στο ίδιο pixel, ο αλγόριθμος του rendering πρέπει να έχει ένα τρόπο να αποφασίσει ποιο από αυτά θα είναι ορατό και ποιο δεν θα είναι. Προς την κατεύθυνση αυτή βοηθάει το z-buffer.

Ο αλγόριθμος συγκρίνει τις τιμές των βαθών των pixels που πρέπει να προβληθούν στο ίδιο σημείο της οθόνης μας και αποφασίζει ποιο αντικείμενο θα είναι ορατό (ζωγραφίζοντας το pixel στα χρώματά του). Δηλαδή αν το pixel θα παραμείνει ως έχει (όταν το ήδη ζωγραφισμένο αντικείμενο είναι πιο μπροστά) ή αν θα αλλάξει σε κάτι άλλο (όταν το ήδη ζωγραφισμένο αντικείμενο είναι πιο πίσω).

Όταν δύο pixels είναι εξαιρετικά κοντά το ένα με το άλλο και μια και η ακρίβεια που μπορούν να έχουν οι αριθμοί κινητής υποδιαστολής (floating point numbers ή floats) στους υπολογιστές είναι πεπερασμένη, υπάρχει πιθανότητα το ένα pixel να ανήκει στην μία επιφάνεια, αλλά το διπλανό του να ανήκει στην άλλη. Ως αποτέλεσμα, το ένα pixel είναι πράσινο ενώ το διπλανό του λευκό. Αν κινήσετε το viewport της σκηνής με το ποντίκι, θα δείτε ότι οι επιφάνειες τρεμοπαίζουν αλλάζοντας χρώμα συνέχεια. Αυτό συμβαίνει μια κι ο αλγόριθμος πέφτει στη λούπα των λαθών αυτών. Το pixel που ανήκει στη μία επιφάνεια είναι πιο πάνω από το αντίστοιχο της άλλης (αφού κι ο float του βάθους έχει υπολογιστεί μ' αυτό τον τρόπο), ενώ το διπλανό του είναι πιο κάτω (οπότε πρέπει να έχει τα χρώματα του pixel της δεύτερης), αφού υπάρχει λάθος στον float του βάθους του.

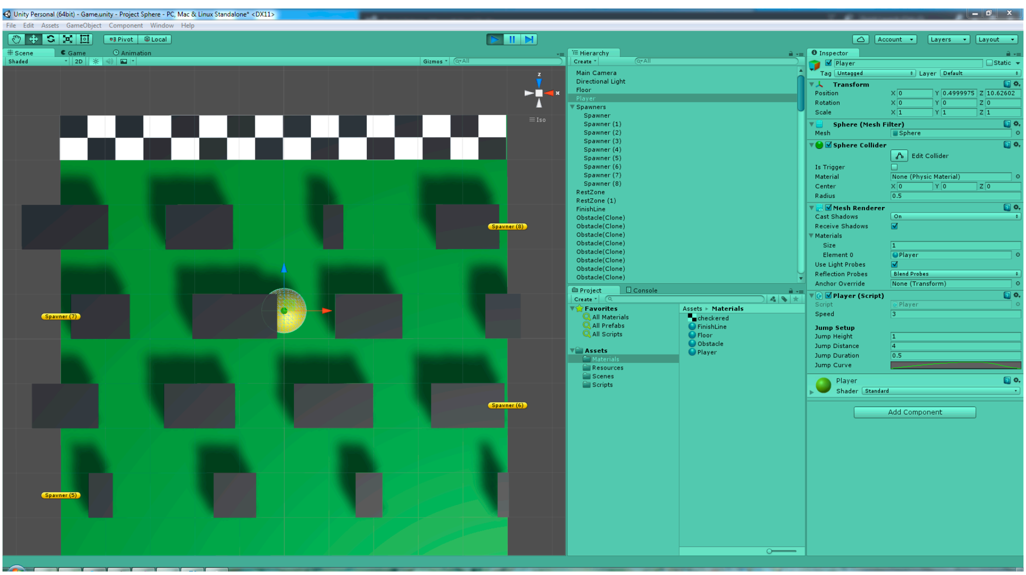
Το συγκεκριμένο θέμα αποτελεί κυρίως πρόβλημα της μηχανής και του τρόπου που κάνει render. Μια και είναι πέρα από τις δυνατότητές μας να το επιλύσουμε στη ρίζα του, θα το ξεπεράσουμε με έναν απλό τρόπο. Θα τοποθετήσουμε λίγο πιο ψηλά τα planes των γραμμών, ώστε να είναι ελάχιστα πιο πάνω από το plane της πίστας, αποφεύγοντας έτσι το πρόβλημα.

Η λύση του προβλήματος

Επιλέξτε τις δύο λευκές και τη μία καρό γραμμές κι αλλάξτε τη θέση τους στον άξονα y, από 0 σε 0.001. Παρατηρήστε ότι αν προσπαθήσετε να τις τοποθετήσετε στο 0.0001, το πρόβλημα εξακολουθεί να υπάρχει. Αυτό έχει να κάνει με το εύρος του σφάλματος που συμβαίνει κατά το rendering. Μετά από αυτή την αλλαγή, οι γραμμές θα πρέπει να είναι ολόκληρες και να καλύπτουν την πίστα από άκρη σε άκρη.

Από τη στιγμή που τώρα φαίνεται καθαρά, σίγουρα θα το παρατηρήσατε. Το texture της γραμμής τερματισμού τεντώνεται ώστε να καλύψει όλη την επιφάνειά της, με αποτέλεσμα να μη φαίνεται όμορφο. Προφανώς όμως θέλουμε να το κάνουμε να φαίνεται σωστά. Πρέπει να ρυθμίσουμε το tiling του material. Το tiling του material υποδηλώνει αν το ίδιο texture θα επαναληφθεί περισσότερες από μία φορές (και σε ποιον άξονα) με σκοπό να καλύψει την επιφάνεια που του έχει ανατεθεί.

Πηγαίνοντας στον Inspector έχοντας επιλέξει το material FinishLine από το παράθυρο του project, στο πεδίο Tiling θέτουμε X=8. Αυτό σημαίνει ότι το texture θα επαναληφθεί 8 φορές στον άξονα X. Πλέον, βλέπουμε ότι το καρό της γραμμής τερματισμού εμφανίζεται όπως πρέπει.



Βλέπουμε ότι η σφαίρα "αποκτά" σφάλμα σε κάθε της άλμα και, λειτουργώντας αθροιστικά, το σφάλμα γίνεται πιο αντιληπτό όσο απομακρυνόμαστε από την αρχή της πίστας.

Κάποιες λεπτομέρειες

Τρέχοντας το παιχνίδι βλέπουμε ότι τα άλματα της σφαίρας του παίχτη είναι πολύ μικρά. Μπορούμε αν θέλουμε να μικρύνουμε όλη την πίστα και ν' αλλάξουμε όλες τις θέσεις από τους spawners, αλλά θα ήταν απλούστερο να αλλάξουμε το Jump Distance του Player script, συγκεκριμένα από 2 σε 4. Κι επειδή πρέπει να πιάσουμε και λίγο κώδικα, για να μην τον ξεχνάμε, πάμε να κάνουμε μερικές αλλαγές στην κίνηση της σφαίρας.

Αν προχωρήσετε τη σφαίρα αρκετά στην πίστα, θα παρατηρήσετε ότι δεν πέφτει ακριβώς πάνω στο κέντρο της γραμμής της πίστας. Αυτό γίνεται και πάλι λόγω rounding errors στον κώδικά μας. Υπολογίζουμε το πόσο μπροστά πρέπει να μετακινηθεί η σφαίρα με βάση τα δευτερόλεπτα που περνάνε από την εκτέλεση του άλματος. Το αποτέλεσμα είναι η σφαίρα να μην πέφτει ακριβώς στην τροχιά που πρέπει, αλλά να είναι λίγο πιο πίσω. Όσο πιο μακριά στην πίστα πάμε, τόσο περισσότερο γίνεται αντιληπτό το φαινόμενο.

Στην περίπτωση αυτή θα διορθώσουμε το bug με μια χακεριά, μια και ξέρουμε ότι το άλμα είναι ελάχιστα ελλιπές στο τέλος της κίνησής του. Θα θέσουμε λοιπόν τη θέση της σφαίρας ίση με τις μονάδες της αυξημένες κατά 0.5, όταν ο κώδικας αποφασίζει ότι το άλμα έχει τελειώσει.

Οπότε ο κώδικας της μεθόδου handleJump του Player script πρέπει να αλλάξει, όπως φαίνεται παρακάτω:

```
private void handleJump()
{
      currentJumpTime += Time.deltaTime;
      float animationEval = JumpCurve.Evaluate(currentJumpTime *
      relativeCurveTime);
      float currentHeight = normalY + (animationEval * JumpHeight);
      float forwardMovement = jumpDirection * JumpDistance * Time.deltaTime;
      Vector3 jumpVector = new Vector3(transform.position.x, currentHeight,
      transform.position.z + forwardMovement);
      transform.position = jumpVector;
      if (currentJumpTime >= JumpDuration)
      {
            transform.position = new Vector3(transform.position.x, transform.
            position.y, Mathf.FloorToInt(transform.position.z) + 0.5f);
            jumping = false;
      }
}
```

Η αλλαγή που κάνουμε είναι απλή. Μόλις ο κώδικας δει ότι το άλμα έχει τελειώσει, θα διορθώσει τη θέση της σφαίρας προσθέτοντας μισή ακόμα μονάδα στις ακέραιες μονάδες της τρέχουσας θέσης του. Πλέον, η σφαίρα μετά από κάθε άλμα της θα τοποθετείται ακριβώς στο κέντρο της τροχιάς της γραμμής που περνάνε τα εμπόδια.

Ακόμα ένα θέμα με την παραπάνω εικόνα είναι και τα εμπόδια που δημιουργούνται. Όταν δημιουργείται ένα εμπόδιο μέσω της Instantiate, τοποθετείται στην ιεραρχία χωρίς να έχει κάποιο άλλο αντικείμενο ως πατέρα. Το αποτέλεσμα είναι να γεμίζει η ιεραρχία από αντικείμενα. Αφού τα αντικείμενα που πέρασαν από την πίστα δεν χρειάζονται πια, θα χρησιμοποιήσουμε collision detection ώστε να τα καταστρέφου-

με από την πίστα. Μόνο που αυτό θα γίνει αφού συζητήσουμε για τη μηχανή φυσικής της Unity. Προς το παρόν θα αλλάξουμε τον κώδικα, ώστε κάθε εμπόδιο που εμφανίζεται στην πίστα να μπαίνει αυτόματα στο Hierarchy, ως παιδί της γεννήτριας που το δημιούργησε. Αλλάζουμε τον κώδικα της μεθόδου spawnObstacle του αρχείου Spawner.cs:

```
private void spawnObstacle()
{
      Obstacle.Instantiate(new InitializationData()
      {
            Position = transform.position,
            Rotation = transform.rotation,
            Prefab = ObstaclePrefab
      }, WidthModifier.Random(), Speed).transform.SetParent(transform);
}
```

Κι εδώ απλή είν' η αλλαγή. Μόλις δημιουργήσουμε το εμπόδιο, ως parent του transform του θέτουμε το transform του αντικειμένου που τρέχει το script (δηλαδή του Spawner).

Σαν μια τελευταία πινελιά καλλωπισμού του Hierarchy του project, σέρνουμε τα δύο RestZones και το FinishLine στο αντικείμενο Floor, ώστε να γίνουν παιδιά του. Στο κάτω κάτω της γραφής, είναι αντικείμενα του δαπέδου και είναι λογικό να υπάρχουν σε αυτό.

Στο παρόν άρθρο, λοιπόν, εργαστήκαμε πάνω στην πίστα. Αρχικά δημιουργήσαμε τις γεννήτριες εμποδίων και τελειοποιήσαμε τις θέσεις τους πάνω στην πίστα. Ομορφύναμε την πίστα και ταυτόχρονα σπάσαμε τη μονοτονία του πράσινου χρώματός της, προσθέτοντας τις λευκές νεκρές ζώνες και τη γραμμή τερματισμού. Αντιμετωπίσαμε το πρόβλημα που είχε η κίνηση της σφαίρας του παίκτη, αλλά και το πρόβλημα του z-fighting των textures που μας παρουσιάστηκε. Τέλος, καλυτερεύσαμε την ιεραρχία των αντικειμένων στην πίστα μας.

Όπως πάντα, μπορείτε να βρείτε την τελευταία έκδοση του κώδικα στο https://github.com/ikromm/project-sphere. Σας παροτρύνουμε να υποβάλετε τυχόν απορίες αλλά και να προτείνετε βελτιώσεις, στο site του περιοδικού.

Πρώτη γνωριμία κι εξάσκηση με το Git

Στο πρώτο μέρος της μίνι σειράς μας περί των συστημάτων διαχείρισης εκδόσεων, ξεκινήσαμε με μια παρουσίαση μερικών εισαγωγικών κι απαραίτητων εννοιών. Συνεχίσαμε εστιάζοντας την προσοχή μας στο Git. Αν και φροντίσαμε να το παινέψουμε αρκετά, η αλήθεια είναι ότι προκειμένου να το εκτιμήσει κάποιος πρέπει να το χρησιμοποιήσει στην πράξη – ή έστω να πειραματιστεί μαζί του. Ευτυχώς, δεν χρειάζεται να 'χει κάποιο μεγαλεπήβολο πρότζεκτ στα σκαριά, ούτε καν κάτι συγκεκριμένο.

του Χρήστου Βαρελά

Όπως μπορείτε να δείτε κι από τον επίσημο δικτυακό τόπο του Git (https://git-scm.com), το δημοφιλές κατανεμημένο Version Control System διατίθεται, μεταξύ άλλων, για διανομές Linux, για το macOS αλλά και για τα Windows. Εναλλακτικά, ο χρήστης έχει τη δυνατότητα να το μεταγλωττίσει ξεκινώντας από τον πηγαίο κώδικα και κατόπιν να το εγκαταστήσει στο λειτουργικό της προτίμησής του. Αν και υπάρχουν γραφικά αλλά και web front-ends για το Git, έχει περισσότερο νόημα να το γνωρίσουμε *πρώτα* από το φυσικό του περιβάλλον. Αυτό δεν είναι άλλο από τη γραμμή εντολών. Αφού κατανοήσουμε τη λειτουργία του από εκεί, μετά δεν θα έχουμε κανένα πρόβλημα με οποιοδήποτε άλλο front-end. Στο παρόν άρθρο δουλεύουμε σε περιβάλλον openSUSE Leap. Περιττό να σημειώσουμε ότι παρόμοια θα εργαστεί κανείς από οποιαδήποτε άλλη διανομή Linux, από το macOS ή ακόμη κι από τα Windows. (Παραδεχόμαστε, ωστόσο, ότι σε περιβάλλον Windows δεν έχουμε χρησιμοποιήσει ποτέ το Git.) Για την εγκατάσταση δεν επιλέξαμε τη μεταγλώττιση από το source. Πολύ πιο απλά, στραφήκαμε στον package manager της διανομής μας:

```
cvar@ohsuse:~> sudo zypper ref
cvar@ohsuse:~> sudo zypper in git-core
```

Σημειώνουμε ότι στα αποθετήρια της διανομής σας πιθανώς να μην υπάρχει η πλέον πρόσφατη έκδοση του Git (δείτε τη στην πρώτη σελίδα του επίσημου δικτυακού τόπου). Ακόμη κι έτσι να 'χουν τα πράγματα, σ' αυτό το στάδιο ποσώς αξίζει ν' ασχοληθούμε.

Τώρα, κάθε φορά που κάνουμε ένα commit (βλ. παρακάτω) στο πρότζεκτ που δου-

λεύουμε, το συνοδεύουμε κι από ένα σύντομο μήνυμα. Επειδή πρέπει να φαίνεται εύκολα ποιοι έχουν κάνει commit στο πρότζεκτ, ορίζουμε εξ αρχής δύο σχετικές παραμέτρους – συγκεκριμένα για το όνομα και για το email μας:

```
cvar@ohsuse:~> git config --global user.name "cvar"
cvar@ohsuse:~> git config --global user.email "cvar@colder.xyz"
```

Οι παράμετροι που ορίζουμε ή θέτουμε για το Git πηγαίνουν στο αρχείο .gitconfig, στον προσωπικό μας κατάλογο. Προς το παρόν έχουμε θέσει δύο παραμέτρους:

```
cvar@ohsuse:~> cat .gitconfig
[user]
      name = cvar
      email = cvar@colder.xyz
```

Αντί να κοιτάζουμε τα περιεχόμενα του αρχείου .gitconfig, κάλλιστα μπορούμε να δίνουμε στο git την εντολή config μαζί με την παράμετρο --config:

```
cvar@ohsuse:~> git config --list
user.name=cvar
user.email=cvar@colder.xyz
```

Αναμενόμενα, υπάρχουν ένα σωρό άλλες παράμετροι που έχουμε την ευκαιρία να θέσουμε. Για παράδειγμα, μερικές φορές το Git θα σταματά ό,τι κάνει και θα περιμένει από εμάς να πληκτρολογήσουμε κάτι. Για editor, θα χρησιμοποιεί τον προκαθορισμένο του συστήματος. Στο δικό μας είναι ο vim:

```
cvar@ohsuse:~> which $EDITOR
/usr/bin/vim
```

Αν προτιμάμε κάποιον άλλον *ειδικά* για το Git, γράφουμε:

```
cvar@ohsuse:~> git config --global core.editor nano
cvar@ohsuse:~> git config --list
user.name=cvar
user.email=cvar@colder.xyz
core.editor=nano
```

Προς το παρόν δεν χρειάζεται ν' ασχοληθούμε άλλο με τις παραμέτρους. Έχουμε το Git εγκατεστημένο και μια χαρά ρυθμισμένο και είναι ώρα ν' αρχίσουμε να το μαθαίνουμε.

Το πρώτο μας αποθετήριο

Ξεκινάμε με ένα απλό, δοκιμαστικό, πρότζεκτ, ώστε να συνηθίζουμε σιγά σιγά το Git και βεβαίως ν' αποκτήσουμε μια καλή ιδέα για το τι μπορεί να κάνει για εμάς. Το πρότζεκτ θα ζει κάτω από τον κατάλογο με όνομα theGitProject, οπότε τον δημιουργούμε και μεταβαίνουμε σ' αυτόν:

```
cvar@ohsuse:~> mkdir theGitProject
cvar@ohsuse:~> cd theGitProject
cvar@ohsuse:~/theGitProject>
```

Προς το παρόν το πρότζεκτ είναι κενό, δεν περιλαμβάνει ούτε ένα αρχείο. Αυτό αλλάζει εύκολα:

```
cvar@ohsuse:~/theGitProject> touch somefile
```

Ωραία, για αρχή το κενό μας αρχείο, ονόματι somefile, είναι αρκετό. Η επόμενη κίνηση είναι να αρχικοποιήσουμε ένα repository (αποθετήριο) για το φιλόδοξο πρότζεκτ μας. Μη γελάτε. Μπορεί να μη δημιουργήσουμε κάποια αξιοπρόσεκτη εφαρμογή ή δικτυακό τόπο, θα σημειώσουμε όμως σημαντική πρόοδο στο σχέδιό μας για την κατάκτηση του Git. Μη λέτε, λοιπόν, ότι δεν είμαστε φιλόδοξοι. Η αρχικοποίηση του repository επιτυγχάνεται έτσι:

```
cvar@ohsuse:~/theGitProject> git init
Initialized empty Git repository in /home/cvar/theGitProject/.git
```

Μάλιστα. Όπως βλέπουμε από το μήνυμα, το νέο repository αρχικοποιήθηκε επιτυχώς και προς το παρόν είναι κενό.

Παρακολούθηση αρχείων

Η δημιουργία repository εντός καταλόγου, δεν σημαίνει ότι το Git παρακολουθεί (tracks) τα αρχεία του ίδιου καταλόγου. Πράγματι:

```
cvar@ohsuse:~/theGitProject> git status
On branch master
```

```
Initial commit
Untracked files:
  (use "git add <file>..." to include in what will be committed)
      somefile
nothing added to commit but untracked files present (use "git add" to track)
```

Το ένα και μοναδικό αρχείο μας, το somefile, είναι, λέει, **untracked**. Μ' άλλα λόγια, δεν παρακολουθείται από το Git. Μπορούμε όμως εύκολα να το υποβάλλουμε σε παρακολούθηση, ώστε το Git να πάψει να το αγνοεί:

```
cvar@ohsuse:~/theGitProject> git add somefile
```

Δεν πήραμε κάποιο μήνυμα στο τερματικό μας – κι αυτό είναι καλό. Αν δώσουμε ξανά git status, διαπιστώνουμε ότι το Git βλέπει, πλέον, το αρχείο somefile και μάλιστα θα το συμπεριλάβει στο επόμενο commit:

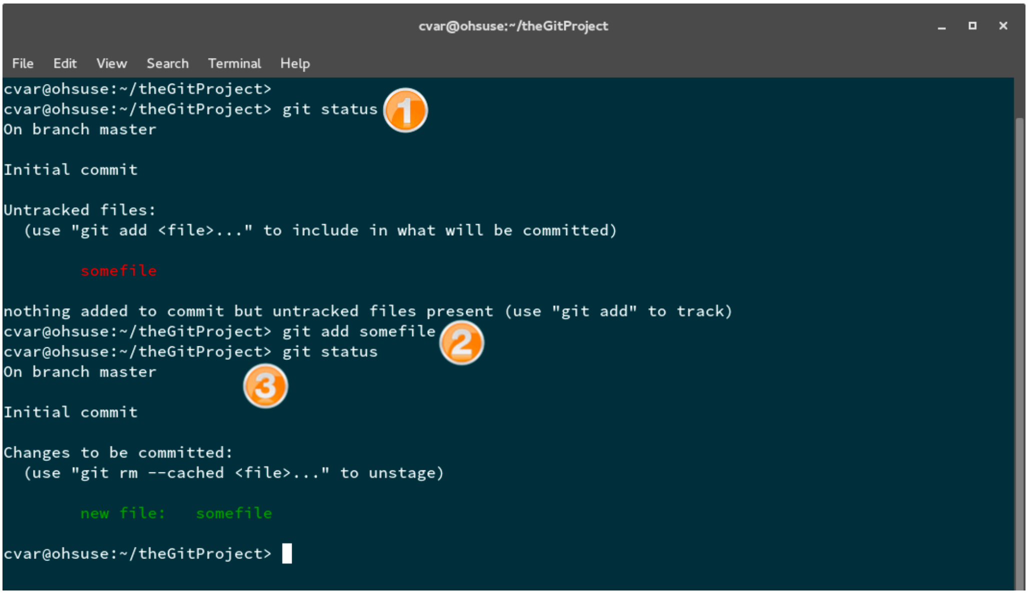
```
cvar@ohsuse:~/theGitProject> git status
On branch master
Initial commit
Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
      new file:    somefile
```

Σημειώστε ότι αν είχαμε πολλά αρχεία και θέλαμε να τα υποβάλλουμε όλα μαζί υπό παρακολούθηση, τότε θα μπορούσαμε να χρησιμοποιήσουμε χαρακτήρες μπαλαντέρ. Για παράδειγμα, με

```
git add *.c
```

ξεκινάμε το tracking όλων των αρχείων με κατάληξη ".c". Για το tracking όλων ανεξαιρέτως των αρχείων του καταλόγου του πρότζεκτ, θα γράφαμε:

```
git add .
```



Το αρχείο somefile μέσα στον κατάλογο του πρότζεκτ μας δεν παρακολουθείται από το Git: είναι untracked (1). Αφού όμως το προσθέσουμε στα αρχεία που επιβλέπει το Git (2), τότε παρακολουθείται και μάλιστα θα συμπεριληφθεί στο επόμενο commit (3).

Το πρώτο commit

Κάθε φορά που προσθέτουμε αρχεία υπό παρακολούθηση ή τροποποιούμε υπάρχοντα αρχεία (πάντα υπό παρακολούθηση), καλό είναι να κάνουμε κι ένα commit. Ειδικά για τον κόσμο του Git –και σε αντίθεση με άλλα VCS–, το **commit** σημαίνει ότι καταγράφουμε στην *τοπική* βάση του αποθετηρίου μας τις όποιες αλλαγές έχουν γίνει στο τρέχον branch (βλ. παρακάτω). Αν μετά θέλουμε να στείλουμε τις αλλαγές σε ένα απομακρυσμένο αποθετήριο, τότε κάνουμε το λεγόμενο **push**. Κάθε push πρέπει να περιλαμβάνει τουλάχιστον ένα commit και κάθε commit πρέπει να αφορά σε τουλάχιστον ένα αρχείο και να συνοδεύεται από ένα μήνυμα. Δεν υπάρχει περιορισμός στο πλήθος των commits, ωστόσο μετά από κάθε "περιστατικό" προτείνεται να προχωράμε σε commit. Τι σημαίνει όμως περιστατικό; Να σας πούμε. Δεν πρόκειται για κάποιον επίσημο όρο. Απλά, όταν στο παρόν κείμενο μιλάμε για περιστατικό έχουμε κατά νου μια σημαντική αλλαγή, όπως, π.χ., η διόρθωση ενός bug, η προσθήκη μιας συνάρτησης σε ένα αρχείο κώδικα, η πληκτρολόγηση μιας εκτενούς παραγράφου ή ενότητας σε post που ετοιμάζουμε κ.ο.κ. Έχουμε αυτή τη στιγμή κάποιο περιστατικό; Βεβαίως κι έχουμε: Ξεκινήσαμε το πρότζεκτ μας με το tracking του αρχείου somefile! Θα συμφωνήσετε, φανταζόμαστε, ότι η ενέργεια αυτή αποτελεί ορόσημο στον αγώνα μας για την κατανόηση του Git, οπότε σίγουρα είναι ώρα για ένα ωραιότατο commit. Μέσα από τον κατάλογο του πρότζεκτ μας, λοιπόν, αρκεί να πληκτρολογήσουμε:

```
cvar@ohsuse:~/theGitProject> git commit
```

Ανοίγει τότε ο προκαθορισμένος text editor (θέσατε την παράμετρο core.editor;) και καλούμαστε να γράψουμε ένα σύντομο μήνυμα που θα συνοδεύει το commit. Εναλλακτικά, θα μπορούσαμε να είχαμε δώσει το μήνυμα ως παράμετρο, από τη γραμμή εντολών:

```
cvar@ohsuse:~/theGitProject> git commit -m "This is my first commit ever!"
```

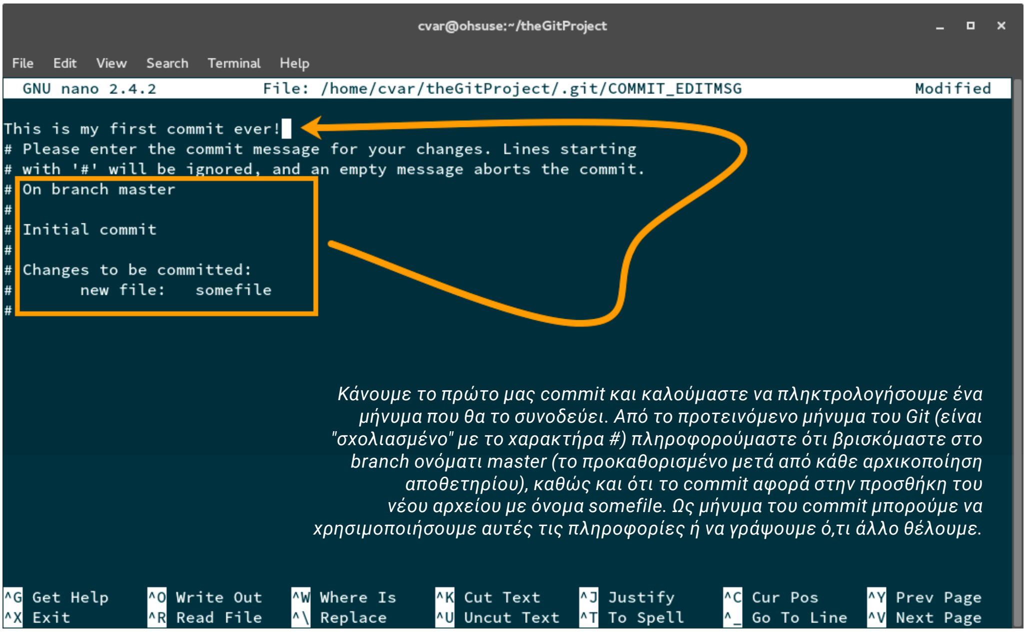
Το commit που μόλις κάναμε αφορούσε σε όλα τα νέα ή/και στα τροποποιημένα –και πάντα υπό παρακολούθηση– αρχεία. Από τη στιγμή που μιλάμε για το πρώτο commit, αυτή η "γενικότητα" δεν πειράζει. Στο εξής όμως τα commits καλό είναι να αφορούν σε συγκεκριμένα αρχεία ή/και σε καταλόγους. Αν, π.χ., θέλουμε ένα commit για μερικές αλλαγές που κάναμε στο αρχείο somefile, τότε γράφουμε:

```
cvar@ohsuse:~/theGitProject> git commit -m "Corrected some typos" somefile
```

Προκειμένου να δούμε όλα τα έως τώρα commits (του τρέχοντος branch), αρκεί να γράψουμε:

```
cvar@ohsuse:~/theGitProject> git log
```

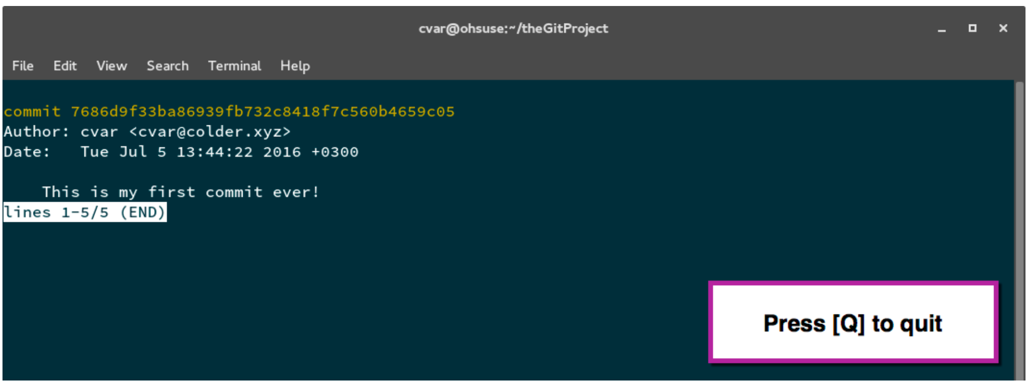
Πατώντας το πλήκτρο [Q] φεύγουμε από τη λίστα με τα commits κι επιστρέφουμε στη γραμμή εντολών.



Κάνουμε το πρώτο μας commit και καλούμαστε να πληκτρολογήσουμε ένα μήνυμα που θα το συνοδεύει. Από το προτεινόμενο μήνυμα του Git (είναι "σχολιασμένο" με το χαρακτήρα #) πληροφορούμαστε ότι βρισκόμαστε στο branch ονόματι master (το προκαθορισμένο μετά από κάθε αρχικοποίηση αποθετηρίου), καθώς και ότι το commit αφορά στην προσθήκη του νέου αρχείου με όνομα somefile. Ως μήνυμα του commit μπορούμε να χρησιμοποιήσουμε αυτές τις πληροφορίες ή να γράψουμε ό,τι άλλο θέλουμε.



Η έξοδος στο τερματικό, μετά το πρώτο μας commit.



Το ένα και μοναδικό commit έως τώρα. Ανά πάσα στιγμή, τα commits του τρέχοντος branch τα βλέπουμε δίνοντας git log. Από τον pager βγαίνουμε πατώντας το πλήκτρο [Q].

Το πρώτο push

Με το Git μπορούμε να εργαζόμαστε αποκλειστικά στον υπολογιστή μας, χωρίς ν' ανεβάζουμε το πρότζεκτ σε απομακρυσμένο server. Ίσως, π.χ., αυτό που θέλουμε είναι μόνο version control στα αρχεία ενός πρότζεκτ μας, οπότε με το Git έχουμε ακριβώς αυτό που χρειαζόμαστε και είμαστε 100% ικανοποιημένοι. Αν όμως συνεργαζόμαστε με άλλους, τότε σίγουρα χρειάζεται ν' ανεβάζουμε τα αρχεία του πρότζεκτ σε server στο τοπικό δίκτυο ή –το πιθανότερο– στο Internet. Στον ίδιο server θ' ανεβάζουν τη δουλειά τους και οι συνεργάτες ή απλά θα μπορούν να παίρνουν τα αρχεία στους δικούς τους υπολογιστές για να κάνουν δοκιμές. Όπως αναφέραμε και νωρίτερα, το ανέβασμα αρχείων ή/και αλλαγών σε απομακρυσμένο Git server –για την ακρίβεια σε απομακρυσμένο repository– ονομάζεται **push**. (Σε άλλα VCS ονομάζεται commit ή checkin, αλλά δεν υπάρχει λόγος να μπερδευόμαστε τώρα.)

Το στήσιμο Git server ξεφεύγει από τους σκοπούς του παρόντος. Μπορούμε πάντως να συνεχίσουμε με τις δοκιμές μας ανοίγοντας έναν δωρεάν λογαριασμό στο ιλιγγιωδώς δημοφιλές GitHub (https://github.com) και δημιουργώντας εκεί ένα δοκιμαστικό, κενό repository. Μόνο προσοχή: τα GitHub repositories για τους δωρεάν

λογαριασμούς είναι δημόσια προσβάσιμα, οπότε προσοχή στο τι ανεβάζετε καθώς δοκιμάζετε (η ομοιοκαταληξία δεν έγινε σκόπιμα). Ας υποθέσουμε ότι το repository του GitHub το έχουμε ονομάσει theProject. Αρχικά, ενημερώνουμε το τοπικό αποθετήριο για την παρουσία του απομακρυσμένου server και τη θέση του αντίστοιχου repository ως εξής:

```
cvar@ohsuse:~/theGitProject> git remote add test-rr git@github.com:colder-is-better/theProject.git
```

Τον απομακρυσμένο server που μόλις προσθέσαμε επιλέξαμε να ονομάσουμε test-rr (από το "test remote repository"). Το username μας στο GitHub, εξάλλου, είναι colder-is-better, οπότε εσείς φροντίστε να βάλετε το δικό σας. Η προηγούμενη εντολή δεν επέστρεψε κάποιο αποτέλεσμα. Γράφοντας όμως git remote -v διαπιστώνουμε ότι η παρουσία του απομακρυσμένου server, ονόματι test-rr, έχει ληφθεί υπόψη.

```
cvar@ohsuse:~/theGitProject> git remote -v
test-rr  git@github.com:colder-is-better/theProject.git (fetch)
test-rr  git@github.com:colder-is-better/theProject.git (push)
```

Πλέον, το push στο αποθετήριο του GitHub με όνομα theProject γίνεται έτσι:

```
cvar@ohsuse:~/theGitProject> git push test-rr master
Counting objects: 3, done.
Writing objects: 100% (3/3), 215 bytes | 0 bytes/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To git@github.com:colder-is-better/theProject.git
 * [new branch]      master -> master
```

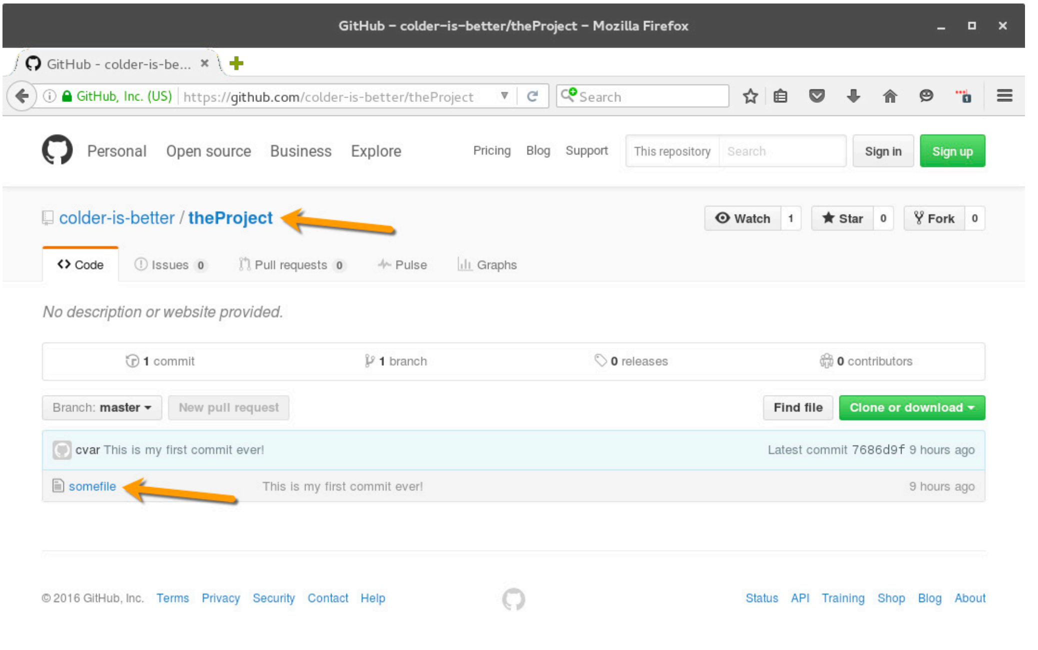
Όλα καλά. Από τον web browser της προτίμησής σας επισκεφτείτε, αν θέλετε, και τη διεύθυνση του repository στο GitHub (π.χ., το σχετικό URL για εμάς είναι το https://github.com/colder-is-better/theProject). Εκεί θα δείτε τα περιεχόμενα του τοπικού repository, το οποίο στο πλαίσιο του παραδείγματός μας προς το παρόν έχει ένα μόνο (κενό) αρχείο: το somefile. Και πριν συνεχίσουμε, δύο μόνο παρατηρήσεις:

- Εξ ορισμού, μετά την αρχικοποίηση ενός τοπικού αποθετηρίου έχουμε ένα μόνο branch: το master. Το περιεχόμενο αυτού στείλαμε στο απομακρυσμένο repository.
- Προκειμένου να δουλέψει το push, στο λογαριασμό μας στο GitHub θα πρέπει να έχουμε στείλει το δημόσιο κλειδί από την τοπική εγκατάσταση του SSH. Περισσότερα για τα SSH passwordless logins και το key-based authentication μπορείτε να διαβάσετε στο https://deltahacker.gr/?p=13357.

Θέλουμε να πιστεύουμε ότι όλα όσα βλέπατε τόσον καιρό στο web ή διαβάζατε απ' εδώ κι απ' εκεί –πάντα περί Git και GitHub–, πλέον έχουν αρχίσει ν' αποκτούν νόημα. Αν μάλιστα έχετε αρχίσει να πωρώνεστε, πραγματικά δεν ξέρουμε τι θα κάνετε στη συνέχεια.



Προσθήκη απομακρυσμένου αποθετηρίου ονόματι theProject από το GitHub (1) – και έλεγχος (2). Μπορούμε πλέον ν' ανεβάσουμε το περιεχόμενο του τοπικού branch, ονόματι master, στο master του απομακρυσμένου repository (3). Ένας οποιοσδήποτε συνεργάτης μας θα είναι σε θέση να κλωνοποιήσει, στον τοπικό του υπολογιστή, το σχετικό αποθετήριο του GitHub (π.χ., git clone git@github.com:colder-is-better/theProject.git).



Ιδού το απομακρυσμένο repository ονόματι theProject, στο GitHub. Στο πλαίσιο των δοκιμών μας περιλαμβάνει ένα και μόνο ένα αρχείο (somefile).

Τα branches και οι φυσικοί νόμοι

Αν δεν ασχοληθούμε με τα branches, δεν πρόκειται να εκτιμήσουμε την αξία του Git. Πρακτικά, το branch είναι μια ξεχωριστή "οικογένεια" αλλαγών επί του περιεχομένου του πρότζεκτ μας. Κάθε αποθετήριο του Git ξεκινά με ένα branch ονόματι master κι επιτρέπεται να έχει περισσότερα του ενός. Σκεφτείτε κάθε branch ως ένα παράλληλο Σύμπαν, το οποίο κάποια δεδομένη χρονική στιγμή –και με τρόπο μυστηριώδη– υφίσταται, αρχίζοντας ακριβώς με ό,τι περιλαμβάνει το master (ή το branch από το οποίο ξεκίνησε, τέλος πάντων). Από εκεί και μετά, το νέο Σύμπαν εκτείνεται και μετασχηματίζεται κι εξελίσσεται με το δικό του τρόπο και ρυθμό, χωρίς φυσικά να επηρεάζει τα άλλα branches.

Σε πολύ λίγο θα φτιάξουμε ένα νέο branch για το δοκιμαστικό μας πρότζεκτ. Σ' αυτό το νέο branch σκοπεύουμε να δουλεύουμε ή/και να δοκιμάζουμε τις ιδέες μας. Όταν θα είμαστε ικανοποιημένοι, θα συγχωνεύουμε (merge) τις προσθήκες ή/και τις αλλαγές που έλαβαν χώρα στο νέο branch, πίσω στο περιεχόμενο του master. Ναι: Στον κόσμο του Git οι φυσικοί νόμοι είναι πιο ελαστικοί σε σύγκριση με τον δικό μας κόσμο –ή τουλάχιστον έτσι υποθέτουμε– κι επιτρέπουν τη μεταφορά και την ενσωμάτωση ύλης μεταξύ παραλλήλων συμπάντων.

Ανά πάσα στιγμή, προκειμένου να δούμε τα branches ενός αποθετηρίου γράφουμε:

```
cvar@ohsuse:~/theGitProject> git branch
* master
```

Όπως βλέπετε κι εσείς, προς το παρόν έχουμε ένα μόνο branch: το master. Αν στην εντολή git branch προσθέσουμε και την παράμετρο -a, βλέπουμε και τα branches στα απομακρυσμένα repositories:

```
cvar@ohsuse:~/theGitProject> git branch -a
* master
  remotes/test-rr/master
```

Και στα δύο παραδείγματα θα παρατηρήσατε φυσικά τον αστερίσκο, ο οποίος υποδεικνύει το branch στο οποίο βρισκόμαστε. Τώρα είμαστε στο master κι αυτό δεν μας προκαλεί καμία έκπληξη. Ας φτιάξουμε επιτέλους ένα νέο branch, ονόματι devel:

```
cvar@ohsuse:~/theGitProject> git checkout -b devel
Switched to a new branch 'devel'
cvar@ohsuse:~/theGitProject> git branch
* devel
  master
```

Μάλιστα. Τι συνέβη μόλις; Να σας πούμε:

- Στην ορολογία του Git, το checkout δεν σημαίνει ό,τι σε άλλα VCS (βλ. πρώτο μέρος της σειράς μας). Αντίθετα, σημαίνει "αλλαγή branch".
- Το checkout μαζί με την παράμετρο -b δημιουργεί ένα νέο branch και ταυτόχρονα μας μεταφέρει σ' αυτό. Στο παράδειγμά μας, λοιπόν, με το git checkout -b devel φτιάχνουμε το νέο Σύμπαν με όνομα devel κι αμέσως τηλεμεταφερόμαστε σ' αυτό.
- Η θέση του αστερίσκου στην έξοδο του git branch έχει φύγει από τ' αριστερά του master και τώρα βρίσκεται σ' αριστερά του devel. Πράγματι, λοιπόν, είμαστε τώρα στο νέο branch, το devel.

Αν για οποιονδήποτε λόγο θέλουμε να πάμε στο master, δίνουμε:

```
cvar@ohsuse:~/theGitProject> git checkout master
Switched to branch 'master'
cvar@ohsuse:~/theGitProject> git branch
  devel
* master
```

Μια χαρά, ας γυρίσουμε όμως και πάλι στο devel:

```
cvar@ohsuse:~/theGitProject> git checkout devel
Switched to branch 'devel'
cvar@ohsuse:~/theGitProject> git branch
* devel
  master
```

Αλλαγές, προσθήκες και συγχώνευση περιεχομένου

Βρισκόμαστε λοιπόν στο branch με όνομα devel, κάτι που φαίνεται κι από την έξοδο της git status:

```
cvar@ohsuse:~/theGitProject> git status
On branch devel
nothing to commit, working directory clean
```

Πάμε αμέσως να προσθέσουμε περιεχόμενο στο –κενό, έως αυτή τη στιγμή– αρχείο, με όνομα somefile. Πληκτρολογούμε:

```
cvar@ohsuse:~/theGitProject> echo "this ain't Kansas no more" >> somefile
```

```
cvar@ohsuse:~/theGitProject> cat somefile
this ain't Kansas no more
```

Ωραία. Προσέξτε τώρα:

```
cvar@ohsuse:~/theGitProject> git status
On branch devel
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)
      modified:   somefile
no changes added to commit (use "git add" and/or "git commit -a")
```

Το Git κατάλαβε ότι το αρχείο somefile τροποποιήθηκε, μας πληροφορεί όμως ότι δεν είναι staged για το επόμενο commit. Θα γίνει staged αν αρχίσουμε να το παρακολουθούμε (track) *και* στο branch που τώρα βρισκόμαστε. Δίνουμε

```
cvar@ohsuse:~/theGitProject> git add somefile
```

και είμαστε έτοιμοι:

```
cvar@ohsuse:~/theGitProject> git status
On branch devel
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
      modified:   somefile
```

Ας κάνουμε και το commit:

```
cvar@ohsuse:~/theGitProject> git commit -m "added some content" somefile
[devel 1453d47] added some content
 1 file changed, 1 insertion(+)
```

Θυμηθείτε, είμαστε στο branch ονόματι devel.

```
cvar@ohsuse:~/theGitProject> git log
commit 1453d47027760fcff2a99004d4f65553c9e0ca4f
Author: cvar <cvar@colder.xyz>
```

```
Date:   Thu Jul 7 11:20:29 2016 +0300

    added some content

commit 668b62748042fd2b372cda18abd463d87fd92cf3
Author: cvar <cvar@colder.xyz>
Date:   Thu Jul 7 09:57:56 2016 +0300

    This is my first commit ever!
```

Το νέο branch, το devel, έχει "κληρονομήσει" το ένα και μοναδικό commit που είχαμε κάνει στο branch με όνομα master. Λογικό κι αναμενόμενο, με βάση όλα όσα έχουμε πει για τα branches. Αλήθεια, πώς έχει η κατάσταση αυτή τη στιγμή στο master; Στοιχηματίζουμε ότι γνωρίζετε, ας πάμε όμως να δούμε.

```
cvar@ohsuse:~/theGitProject> git checkout master
Switched to branch 'master'
cvar@ohsuse:~/theGitProject> cat somefile
cvar@ohsuse:~/theGitProject> git log
commit 668b62748042fd2b372cda18abd463d87fd92cf3
Author: cvar <cvar@colder.xyz>
Date:   Thu Jul 7 09:57:56 2016 +0300

    This is my first commit ever!
```

Ό,τι περιμέναμε: Στο master δεν υπάρχει καμία αλλαγή στο αρχείο somefile, ενώ αντί για δύο έχουμε ένα μόνο commit. Πιστεύουμε ότι ήδη έχει αρχίσει να φαίνεται η τρομερή ευελιξία που προσφέρουν τα branches του Git. Πάμε πάλι στο devel:

```
cvar@ohsuse:~/theGitProject> git checkout devel
Switched to branch 'devel'
```

Ας δημιουργήσουμε ένα νέο αρχείο, το newdirection:

```
cvar@ohsuse:~/theGitProject> touch newdirection
cvar@ohsuse:~/theGitProject> git status
On branch devel
Untracked files:
  (use "git add <file>..." to include in what will be committed)
```

```
      newdirection
nothing added to commit but untracked files present (use "git add" to track)
cvar@ohsuse:~/theGitProject> git add newdirection
cvar@ohsuse:~/theGitProject> git status
On branch devel
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)
      new file:   newdirection
```

Αμέσως μετά τη δημιουργία του, το αρχείο newdirection δεν παρακολουθείται από το git (είναι untracked). Αυτό αλλάζει εύκολα με ένα git add newdirection. Με αφορμή την παρουσία του newdirection (δεν μπαίνουμε στον κόπο να του προσθέσουμε περιεχόμενο τώρα), κάνουμε κι άλλο ένα σχετικό commit:

```
cvar@ohsuse:~/theGitProject> git commit -m "added a new file, will need it
later" newdirection
[devel ff0d768] added a new file, will need it later
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 newdirection
cvar@ohsuse:~/theGitProject> git log
commit ff0d76845f4e8d03a5403bf0ec6d350d5b761f2f
Author: cvar <cvar@colder.xyz>
Date:   Thu Jul 7 11:54:45 2016 +0300

    added a new file, will need it later

commit 1453d47027760fcff2a99004d4f65553c9e0ca4f
Author: cvar <cvar@colder.xyz>
Date:   Thu Jul 7 11:20:29 2016 +0300

    added some content

commit 668b62748042fd2b372cda18abd463d87fd92cf3
Author: cvar <cvar@colder.xyz>
Date:   Thu Jul 7 09:57:56 2016 +0300

    This is my first commit ever!
```

Μια χαρά, στο devel έχουμε τρία commits. Πάμε *ξανά* να δούμε πώς έχει η κατάσταση στο branch ονόματι master:

```
cvar@ohsuse:~/theGitProject> git checkout master
Switched to branch 'master'
cvar@ohsuse:~/theGitProject> ls -lh
total 0
-rw-r--r-- 1 cvar users 0 Jul  7 11:57 somefile
cvar@ohsuse:~/theGitProject> git log
commit 668b62748042fd2b372cda18abd463d87fd92cf3
Author: cvar <cvar@colder.xyz>
Date:   Thu Jul 7 09:57:56 2016 +0300

    This is my first commit ever!
```

Θεσπέσια όλα! Η θεωρία συμβαδίζει με την πράξη: Το αρχείο newdirection πολύ σωστά δεν υπάρχει στο master, ενώ επίσης πολύ σωστά έχουμε ένα μόνο commit. Έχει φτάσει η στιγμή να κάνουμε την πρώτη μας συγχώνευση (merge). Θα φέρουμε, μ' άλλα λόγια, όλες τις αλλαγές και τις προσθήκες που έχουν ήδη γίνει στο devel, πίσω στο master. Κάτι που πρέπει να θυμόμαστε πριν το merge, είναι ότι πρέπει να βρισκόμαστε στο branch όπου πρόκειται να εισαχθούν οι αλλαγές. Στο πλαίσιο του παραδείγματός μας, είμαστε ήδη σ' αυτό (είναι το master). Για το merge αρκεί να πληκτρολογήσουμε

```
cvar@ohsuse:~/theGitProject> git merge devel
Updating 668b627..ff0d768
Fast-forward
 newdirection | 0
 somefile     | 1 +
 2 files changed, 1 insertion(+)
 create mode 100644 newdirection
```

Όλα δείχνουν ότι το merge πραγματοποιήθηκε επιτυχώς, όμως ας κάνουμε και τους δικούς μας ελέγχους:

```
cvar@ohsuse:~/theGitProject> ls -lh
total 4.0K
-rw-r--r-- 1 cvar users  0 Jul  7 12:14 newdirection
-rw-r--r-- 1 cvar users 26 Jul  7 12:14 somefile
cvar@ohsuse:~/theGitProject> cat somefile
```

```
this ain't Kansas no more
```

Υπάρχει πλέον το αρχείο newdirection, ενώ και το somefile δεν είναι κενό. Επιτυχία! Α, και μην ξεχνάτε και τα commits που έγιναν στο devel:

```
cvar@ohsuse:~/theGitProject> git log
commit ff0d76845f4e8d03a5403bf0ec6d350d5b761f2f
Author: cvar <cvar@colder.xyz>
Date:   Thu Jul 7 11:54:45 2016 +0300

    added a new file, will need it later

commit 1453d47027760fcff2a99004d4f65553c9e0ca4f
Author: cvar <cvar@colder.xyz>
Date:   Thu Jul 7 11:20:29 2016 +0300

    added some content

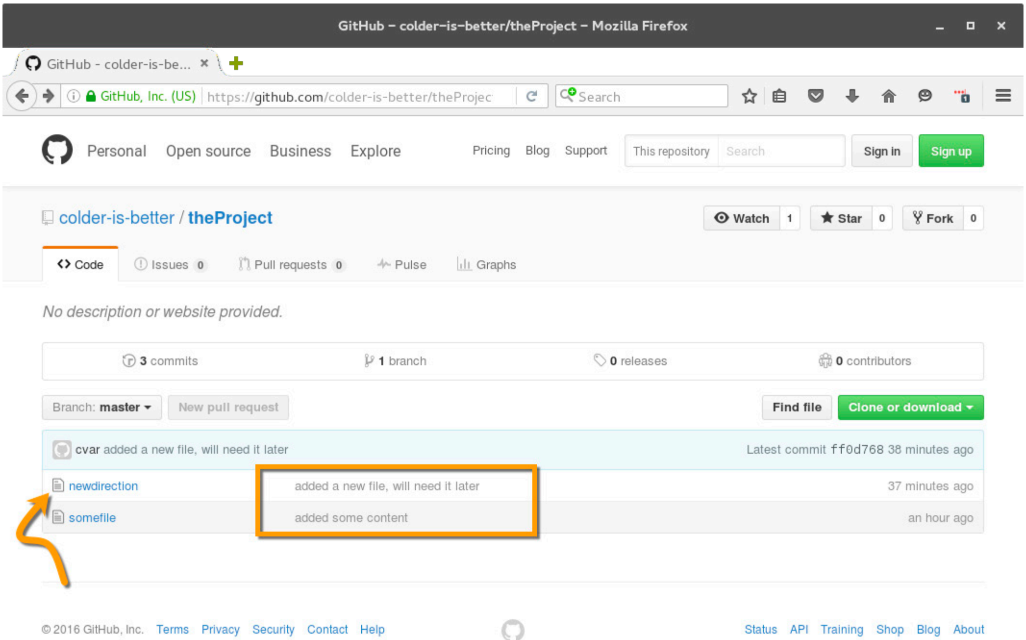
commit 668b62748042fd2b372cda18abd463d87fd92cf3
Author: cvar <cvar@colder.xyz>
Date:   Thu Jul 7 09:57:56 2016 +0300

    This is my first commit ever!
```

Θυμόσαστε εκείνο το απομακρυσμένο repository, στο GitHub; Κρίμα δεν είναι να μην περιλαμβάνει τις αλλαγές που έχουν συντελεστεί στο (τοπικό) master; Πράγματι είναι κρίμα, γι' αυτό ας στείλουμε τις αλλαγές:

```
cvar@ohsuse:~/theGitProject> git push test-rr master
Counting objects: 5, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 510 bytes | 0 bytes/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To git@github.com:colder-is-better/theProject.git
   668b627..ff0d768  master -> master
Branch master set up to track remote branch master from test-rr.
```

Από έναν web browser, πηγαίνετε αν θέλετε στο URL του απομακρυσμένου repository και βεβαιωθείτε ότι οι προσθήκες/αλλαγές έχουν ανέβει.



Μετά το push στο GitHub, τα περιεχόμενα του απομακρυσμένου αποθετηρίου έχουν αυξηθεί κατά 100% (μιλάμε, δηλαδή, για 2 αρχεία αντί για 1 :D) Παρατηρήστε και τα μηνύματα των commits, τα οποία κάναμε τοπικά.

Η συνέχεια

Θέλουμε να πιστεύουμε ότι τα VCS γενικότερα –και βεβαίως το Git ειδικότερα– έχουν κερδίσει το ενδιαφέρον σας. Τα δύο άρθρα του μίνι αφιερώματός μας δεν είναι παρά μια εισαγωγή σ' αυτόν τον συναρπαστικό κόσμο, με τα εξαιρετικά χρήσιμα εργαλεία που αλλάζουν και βελτιώνουν τον τρόπο εργασίας μας. Θα έχουμε την ευκαιρία να επανέλθουμε στο Git, ξεκινώντας με τη δημιουργία ενός static site που θα τελεί υπό την εποπτεία του δημοφιλούς αυτού κατανεμημένου VCS. Εσείς, βέβαια, δεν χρειάζεται να μας περιμένετε για να ξεκινήσετε με τους πειραματισμούς σας. Στοιχηματίζουμε μάλιστα ότι αρκετοί θα σπεύσετε να φέρετε το Git στο καθημερινό σας workflow, όπως λένε και στο Λευκαντί Ευβοίας.

Καλή σας συνέχεια – και καλή σας διασκέδαση.

deltahacker.gr